universitätbonn

Testen von kryptographischer Hardware

von

Maximilian Häring

Institute of Computer Science IV
Work Group IT Security
1. Prüfer: Prof. Dr. Michael Meier
2. Prüfer: Dr. rer. nat. Matthias Frank
Betreuer: Dipl. Inf. Matthias Wübeling
Rheinische Friedrich-Wilhelms-Universität Bonn

Datum: 11.05.2015

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Bachelorarbeit selbstständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bonn, 11.05.2015

Maximilian Häring

Zusammenfassung

Diese Arbeit befasst sich mit dem Testen von kryptographischer Hardware. Das Ziel ist es, eine Möglichkeit zu schaffen kryptographische Hardware automatisiert zu testen. Die so entstehenden Ergebnisse sollen unter anderem einen Vergleich unter kryptographischer Hardware ermöglichen.
Um dies zu erreichen, wird zuerst der Begriff der kryptographischen Hardware geklärt und einige mögliche Ausführungen dieser aufgezeigt. Danach werden verschiedene Softwareschnittstellen für kryptographische Hardware und Projekte, die mit diesen arbeiten, vorgestellt.
Anhand des PKCS #11 Standard wird eine Software entworfen und entwickelt, die Token, die die aus PKCS #11 hervorgehende Schnittstelle Cryptoki unterstützen, testet. Mit zwei Geräten wird diese Software evaluiert und Potentiale für diese aufgezeigt.

Abstract

This thesis deals with testing kryptographic hardware. The goal is to create a way to test this kind of hardware in an automatic way. The results from the tests should make it possible to compare cryptographic hardware.
To achieve this, cryptographic hardware is defined and different types of constructions are presented. Then projects and application program interfaces around cryptographic hardware are investigated.
A Software is designed and implemented to test cryptographic hardware that is working with Cryptoki, the API that resuslts from the PKCS #11 standard. Two devices are used to evaluate this Software and show potential of future works in this area.

Inhaltsverzeichnis

1 Motivation

Im Laufe der letzten Jahre stand das Thema Sicherheit in der IT immer wieder im Fokus der Medien. Sei es durch die Enthüllungen von Whistleblowern wie Snowden[32] und Manning[17] oder durch die regelmäßig beschriebenen Bedrohungen der Unternehmen durch Wirtschaftsspionage[33].
Um Ihre Geschäftsdaten zu schützen, können Unternehmen zu kryptographischer Hardware greifen. Das können Smartcards sein, auf denen ein privater Schlüssel gespeichert ist oder Hardware für den PC, die bestimmte Algorithmen besonders effizient und sicher ausführen. Es gibt viele Geräte mit kryptographischen Funktionen. In ihrer Form und ihrem Funktionsumfang sind sie allerdings sehr unterschiedlich. Ein Datenblatt des Gerätes verrät leider oft nur oberflächliche Informationen.

Möchte ein Unternehmen seine Mitarbeiter mit kryptographischen Schlüsseln ausstatten, um beispielsweise eine Anmeldung am Rechner zu ermöglichen, steht es vor der Wahl sich zwischen unterschiedlichen Geräten zu entscheiden.
Die Entscheidung wird beeinflusst durch den erwünschten Funktionsumfang, aber möglicherweise auch durch die Performance der Hardware und die Möglichkeiten der Anbindung an vielleicht bestehende oder neue Software.
In dieser Arbeit werden verschiedene Arten von kryptographischer Hardware und deren Anbindungsmöglichkeiten an Software vorgestellt. Mit diesen Informationen wird ein Entwurf erstellt. Ziel ist es, eine Software zu schreiben, die einen Teil zu der Entscheidung beitragen kann, welche kryptographische Hardware verwendet wird. Dies soll anhand von Tests geschehen, die nähere Informationen über den tatsächlichen Funktionsumfang und deren Performance geben. So wird eine herstellerunabhängige Möglichkeit geschaffen, Teile der Funktionen eines Tokens in Erfahrung zu bringen.

2 Grundlagen

Dieses Kapitel legt die erarbeiteten und recherchierten Grundlagen für diese Bachelorarbeit dar. Das Kapitel dient weiterhin dazu, Begriffe zu definieren und zu erläutern.

2.1 Begriffe

Kryptographisches Gerät:
Ein Hilfsmittel das kryptographische Informationen speichern und optional auch kryptographische Funktionen ausführen kann. Dieses Hilfsmittel kann sowohl in Hardware (kryptographische Hardware) als auch in Software oder durch eine Kombination implementiert sein. [28]

Hardware security module:
Der Begriff "HSM" (hardware security module) ist nicht eindeutig definiert. Es gibt Werke in denen er als eine kryptographische Hardware definiert ist und für gewöhnlich in einem PC oder Server verbaut ist [1].
In anderen Quellen wird auch tragbare kryptographische Hardware zu diesen gezählt [15].
Gemeinsam haben die Definitionen, dass HSMs kryptographische Hardware sind und so in verschiedenen Anwendungsfällen verwendet werden können, in denen kryptographische Funktionen gebraucht werden. Das kann unter anderem die Beschleunigung der Generierung von Zufallszahlen oder die Ausführung kryptographischer Funktionen sein. Aber auch die Aufbewahrung von kryptographischen Schlüsseln. Im Rahmen dieser Arbeit werden die HSMs als kryptographische Hardware, die für gewöhnlich in Rechnern fest verbaut sind, bezeichnet, um sie von tragbaren kryptographischen Geräten abzugrenzen.

2.2 Integrated circuit card

Die "integrated circuit card", auch Smartcard genannt, wird in ISO/IEC 7816 Teil 1 [12] bis Teil 15 [13] als Standard beschrieben und ist eine kryptographische Hardware. Smartcards können sowohl zur Authentifikation genutzt werden, als auch zur Ausführung von kryptographischen Funktionen [9].
Smartcards werden in zwei Arten unterteilt: die kontaktbehafteten und die kontaktlosen.

Bei Ersteren funktioniert die Kommunikation über Kontakte auf der Karte. Kontaktlose Smartcards agieren mit Techniken wie NFC. Die folgenden Ausführungen sind bezogen auf Smartcards mit Kontakten, lassen sich aber auch auf jene ohne Kontakt übertragen. Die ersten drei Teile des Standards beziehen sich auf kontaktbehaftete Smartcards. Die restlichen Teile definieren Eigenschaften die unabhängig von der physikalischen Schnittstelle sind. [6]

Der zweite Teil [14] spezifiziert Ausmaße und Position der Kontakte auf einer ID-1 Karte. Eine ID-1 Karte wird als eine Karte mit 85,60 mm x 53,98 mm definiert [11]. Meist unterhalb der Kontakte liegt der Chip, durch den die "integrated circuit card" ihren Namen erhält [31]. Die Kontakte, wie in Abbildung 1 dargestellt, dienen zur

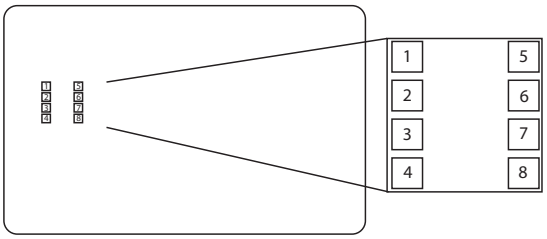

Abbildung 1: *Schematischer Aufbau einer Smartcard mit acht Kontakten*

Kommunikation mit dem Chip auf der Karte. Um ein einheitliches Lesen mit Hilfe eines Lesegerätes über USB gewährleisten zu können existiert das CCID Protokoll [4].

2.3 Softwareschnittstellen

In diesem Abschnitt werden Softwareschnittstellen für kryptographische Hardware vorgestellt.

2.3.1 Microsoft Cryptographic Application Programming Interface

Das "Cryptographic Application Programming Interface" (kurz CryptoAPI), ist ein Service für auf Windows basierende Anwendungen [3]. Langfristig soll dieser durch "Cryptography API: Next Generation" abgelöst werden [5].
CryptoAPI arbeitet mit sogenannten Cryptographic Service Providern (kurz CSP) [7]. Microsoft stellt einige CSPs zur Verfügung [8]. Weitere CSPs können installiert werden, indem eine dynamisch gelinkte Bibliothek (DLL) und eine Signaturdatei bereitgestellt werden [7]. Die Bibliothek ist die Umsetzung der hinzuzufügenden Funktionen. Die Signaturdatei dient dazu, dass die CryptoAPI den CSP erkennt. Auch wird über die Signatur in Intervallen sichergestellt, dass die Bibliothek nicht verändert wurde.
Die Microsoft CSPs stellen Funktionen wie das Ver- und Entschlüsseln und Signieren von Daten mittels des RSA Algorithmus oder das Hashen mittels Algorithmen, wie dem MD5 oder dem SHA zur Verfügung [2].

2.3.2 PKCS#11

PKCS ist eine Reihe von Public-Key Kryptographie Standards die von RSA Laboratories entwickelt und veröffentlicht wurden [30]. Es gibt insgesamt zwölf verschiedene Standards, die unter die Bezeichnung "PKCS" fallen. In Tabelle 1 sind diese aufgeführt. Im Rahmen dieser Arbeit wird, wenn nicht anders vermerkt auf PKCS #11 v2.20 Bezug genommen. Zum aktuellen Zeitpunkt ist die Version 2.4 in Bearbeitung [21].

Bezeichnung	Inhalt
PKCS#1	Implementationsempfehlungen für Public-Key Kryptographie basierend auf dem RSA Algorithmus.
PKCS#3	Implementationsmethode für den Diffie-Hellmann Schlüsselaustausch zum Erzeugen einer sicheren Kommunikation
PKCS#5	Empfehlungen für die Implementation von passwortbasierender Kryptographie.
PKCS#6	Erweiterter Syntax für Zertifikate nach X.509[35].
PKCS#7	Allgemeiner Syntax für Daten die kryptographische Bedeutung haben könnten.
PKCS#8	Syntax für private Schlüssel.
PKCS#9	Beschreibt einige Attribut-Typen die in PKCS #6, PKCS #7, PKCS #8 und PKCS #19 verwendet werden.
PKCS#10	Syntax zum Anfordern der Zertifizierung von einem öffentlichen Schlüssel, einem Namen und möglichen Attributen.
PKCS#11	Beschreibt eine API für Geräte, die kryptographische Informationen speichern oder kryptographische Funktionen ausführen können.
PKCS#12	Beschreibt ein Format zum Speichern z.B. von privaten Schlüsseln und Zertifikaten.
PKCS#13	Beschreibt die Nutzung von elliptischen Kurven zur Kryptographie.
PKCS#15	Beschreibt einen Standard zum Identifizieren mit kryptographischen Token. ISO/IEC 7816-15 ist eine Weiterentwicklung dieser Arbeit.

Tabelle 1: *PKCS Übersicht nach [30]*

Der PCKS #11 Standard [30] beschreibt eine Schnittstelle für kryptographische Hardware, die sowohl von der Plattform als auch der physikalischen Schnittstelle unabhängig ist. Diese Schnittstelle wird Cryptoki genannt und ist in der Programmiersprache C geschrieben.
PKCS #11 wird vom OASIS PKCS #11 Technical Committee weiterentwickelt [21].
Cryptoki kann sowohl auf dem Rechner des Anwenders (dann Softtoken genannt), als auch auf dedizierter Hardware implementiert sein. Handelt es sich um eine Harwareimplementation muss vom Hersteller ein entsprechender Treiber mitgeliefert werden. Dieser Treiber wird im weiteren Verlauf der Arbeit "Modul" genannt. Abbildung 2 stellt den Kommunikationsweg vom Aufruf der Applikation bis zur tatsächlichen Ausführung dar.

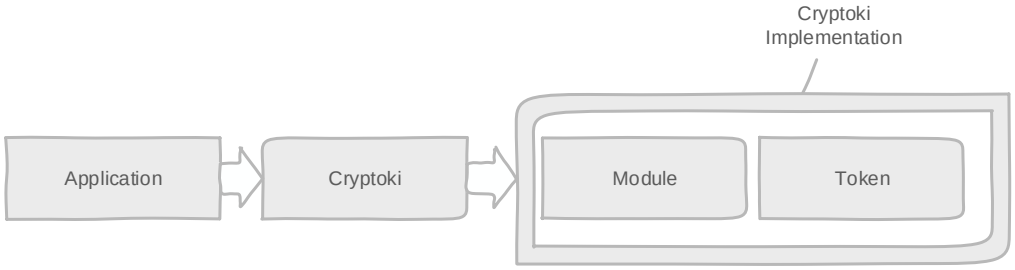

Abbildung 2: *Aufbau des Kommunikationsweges mit Cryptoki*

Funktionskategorien	Anzahl an Funktionen
general-purpose functions	4
slot and token management functions	9
session management functions	8
object management functions	9
encryption functions	4
decryption functions	4
message digesting functions	5
signing and MACing functions	6
functions for verifying signatures and MACSs	6
dual-function cryptographic functions	4
key management functions	5
random number generation functions	2
parallel function management functions	2

Tabelle 2: *PKCS #11 Funktionskategorien [28]*

Funktionsnamen in Cryptoki beginnen mit einem *C_* und sind im CamelCase Format. Der Rückgabewert ist nicht das Ergebnis des Funktionsaufrufes, sondern ein festgelegter Zahlenwert, der Aufschluss über den Erfolg des Aufrufes geben soll. Wenn eine Funktion ein Ergebnis berechnen soll, kann dieses erreicht werden, indem ein entsprechender Buffer der Funktion mit übergeben wurde. Der Lesbarkeit wegen werden in dieser Arbeit wie auch im Standard die Namen der Konstanten verwendet, wenn Rückgabewerte referenziert werden, die in der Schnittstelle verwendet werden, anstatt deren Zahlenwert.
Erkennbar sind diese am Präfix *CKR_*. Eine Übersetzungstabelle befindet sich im Anhang 1. Im Erfolgsfall erhält man den Rückgabewert *CKR_OK*. Dies gilt für alle Funktionen. Für jede Funktion sind die möglichen Rückgabewerte definiert.
Verwendet die Funktion eine Session, so gibt es die Möglichkeit bei einem fehlerhaften Aufruf in dessen Feld *ulDeviceError* Näheres in Erfahrung zu bringen. Der darin enthaltene Fehlercode ist geräteabhängig. [28]

Wenn die Funktion ein Ergebnis liefern soll, übergibt man einen Zeiger auf einen entsprechenden Buffer, der von der Funktion gefüllt wird.
Da die Größe des Rückgabewertes nicht immer vor dem Aufruf festgelegt ist, gibt es eine Konvention: Die Funktion wird einmal mit einem Null-Zeiger als Buffer aufgerufen und die Funktion füllt ein anderes Argument mit der Größe. Danach kann man die Funktion mit einem genügend großen Buffer aufrufen. Auch wenn die Funktion im Programmcode somit zweifach aufgerufen wird, wird diese intern nicht zweimal ausgeführt. Die zurückgegebene Größe ist ein Schätzwert, der nicht viel höher als das richtige Ergebnis sein sollte. [28]

Der folgende Abschnitt geht auf die einzelnen Funktionskategorien ein und beschreibt die in diesen aufkommenden Eigenschaften der Schnittstelle. Tabelle 2 gibt eine Übersicht über diese Kategorien. Der Übersicht wegen wurden die Funktionskategorien erneut gruppiert. Die Gruppierung ist dabei thematisch motiviert.

Allgemeine Funktionen

C_Initialize	*C_Finalize*	*C_GetInfo*
C_GetFunctionList		

Tabelle 3: *PKCS#11: Allgemeine Funktionen*

Die "general-pupose functions" beinhalten Funktionen, siehe Tabelle 3, zum initialisieren (*C_Initialize*) bzw. beenden (*C_Finalize*) der Cryptoki Implementation. *C_Initialize* muss vor jedem weiteren Funktionsaufruf, mit der Ausnahme von *C_GetFunctionList*, ausgeführt werden. Die Funktion *C_GetFunctionList* liefert Zeiger auf alle Funktionen der Cryptoki. Die Funktion *C_GetInfo* füllt eine Struktur, welche dann allgemeine Informationen wie die Versionsnummer zur Schnittstelle enthält. [28]

Slot und Token Verwaltung

C_GetSlotList	*C_GetSlotInfo*	*C_GetTokenInfo*
C_WaitForSlotEvent	*C_GetMechanismList*	*C_GetMechanismInfo*
C_InitToken	*C_InitPIN*	*C_SetPIN*

Tabelle 4: *PKCS#11: Slot und Token Verwaltung*

Ein Slot ist ein logischer Leser, der ein Token enthalten könnte. Ein Token ist hier wiederrum die Abstraktion des kryptographischen Gerätes und wird im Rahmen dieser Arbeit gleichbedeutend genannt. Ein Token muss nicht ein Hardwaretoken sein. Es kann auch ein Token in Software implementiert sein. Ebenso kann ein physikalisches Gerät auch mehrere Token beinhalten.
Ein Mechanismus ist ein Zahlenwert, der eine kryptographische Funktion repräsentiert. Cryptoki unterscheidet zwischen zwei Nutzertypen: dem sogenannten Security Officer

(kurz SO) und dem User. Der Security Officer hat die Berechtigung, den Token zu initialisieren und die User PIN einzustellen.
Die Funktionskategorie "slot and token management functions", siehe Tabelle 4, beinhaltet hauptsächlich Funktionen, um Metadaten über die verwendete Implementation und den Token zu erhalten. Mit der Funktion *C_GetSlotList* erhält man eine Liste aller Slot IDs. Über *C_GetSlotInfo* bzw. *C_GetTokenInfo* erhält man weitere Informationen zu den Slots bzw. Token. *C_WaitForSlotEvent* ist eine Funktion, die auf ein Event eines Slots wartet. Diese Events sind nicht detailliert definiert. Es ist aber damit zu rechnen, dass beim Ein- und Ausstecken eines Token ein Event ausgelöst wird.
C_GetMechanismList gibt eine Liste der unterstützten Mechanismen zurück und mit *C_GetMechanismInfo* lassen sich weitere Informationen über diese erlangen.
Mit *C_InitToken* lässt sich das Token initialisieren. Dabei werden alle Objekte auf dem Token gelöscht und es wird ein Label übergeben. Bis *C_InitPIN* vom SO aufgerufen und somit die User PIN festgelegt wird, ist der Token vom User nicht verwendbar. *C_SetPIN* ändert das Passwort des aktuell eingeloggten Nutzers -also entweder vom User oder vom SO. [28]

Session Funktionen

C_OpenSession	*C_CloseSession*	*C_CloseAllSessions*
C_GetSessionInfo	*C_GetOperationState*	*C_SetOperationState*
C_Login	*C_Logout*	

Tabelle 5: *PKCS#11: Session Funktionen*

Eine Session ist eine logische Verbindung zwischen der Anwendung und dem Token. Sie kann entweder eine rein lesende oder eine Verbindung mit Lese- und Schreibrechten sein. Wenn kein Nutzer eingeloggt ist, handelt es sich um eine öffentliche, nach einloggen -je nach Nutzertyp- um eine User Session oder eine Security Officer Session. Abhängig vom Nutzertypen kann man mit Hilfe der Session auf unterschiedliche Daten zugreifen.
Die Tabelle 5 zeigt die acht Funktionen der "session management functions" Kategorie. Mit *C_OpenSession* öffnet man eine Session. Die Anzahl der möglichen offenen Sessions ist durch die Implementierung der Cryptoki begrenzt. Man kann mit *C_CloseSession* eine bestimmte oder mit *C_CloseAllSessions* alle offenen Sessions schließen. Mit *C_GetSessionInfo* erhält man Informationen wie die Berechtigung über eine bestimmte Session. Mit *C_Login* kann der Nutzer sich einloggen und mit *C_Logout* ausloggen. Der Loginstatus ist sessionübergreifend. Man kann also, soweit es das Token zulässt, mit mehreren Sessions angemeldet sein und alle mit einem *C_Login* Aufruf einloggen.
Mit *C_GetOperationState* und *C_SetOperationState* schafft Cryptoki einen Mechanismus den aktuellen Stand einer Session mit den aktuell laufenden Operationen zu speichern und zu einem späteren Zeitpunkt wieder auszuführen. [28]

Objektverwaltung

C_CreateObject	*C_CopyObject*	*C_DestroyObject*
C_GetObjectSize	*C_GetAttributeValue*	*C_SetAttributeValue*
C_FindObjectsInit	*C_FindObjects*	*C_FindObjectsFinal*

Tabelle 6: *PKCS#11: Objektverwaltung*

Ein Objekt ist eine auf dem Token gespeicherte Information. Objekte lassen sich auf drei Arten klassifizieren: über die Lebensdauer, die Sichtbarkeit und die Objektart.
Es gibt drei Objektarten: data, certificate und key. Diese unterscheiden sich durch ihre Attribute und somit in Ihrem Verwendungszweck. Auch Objekte gleicher Art können weitere unterschiedliche Attribute besitzen. Diese sind vom Mechanismus abhängig.
Ein "data-Objekt" ist eine auf dem Token gespeicherte Information ohne implizite kryptographische Bedeutung.
In einem "certificate-Objekt" kann ein öffentlicher Schlüssel gespeichert werden. Unterstützt werden z.B. X.509 und WTLS Zertifikate.
Sowohl beim data- als auch beim certificate-Objekt sind im Gegensatz zum key-Objekt keine weiteren Abhängigkeiten definiert. Abhängig von der Art des Schlüssels (geheim, privat oder öffentlich), ist es nicht möglich, alle Werte des Objektes auszulesen. Abhängig vom Schlüsseltyp, sind auch beim key-Objekt unterschiedliche Attribute möglich.
Es gibt zwei Arten der Lebensdauer, zwischen denen unterschieden werden muss: Sessionobjekte und Tokenobjekte. Sessionobjekte werden automatisch gelöscht, sobald die Session geschlossen wurde. Tokenobjekte bleiben darüber hinaus bestehen.
Die Sichtbarkeit von Objekten ist in öffentliche und private Objekte zu unterteilen. So wird festgelegt, auf was das Programm zugreifen darf. Öffentliche Objekte lassen sich ohne einen Login verwenden. Private Objekte werden nur sichtbar, wenn man eingeloggt ist.
Mit den Funktionen der "object management functions", siehe Tabelle 6, lassen sich die genannten Objekte erstellen (*C_CreateObject*), kopieren (*C_CopyObject*) und zerstören (*C_DestroyObject*). Objekte enstehen zudem bei den Funktionen *C_GenerateKey*, *C_GenerateKeyPair*, *C_UnwrapKey*, und *C_DeriveKey*.
Mit *C_GetObjectSize* lässt sich die Größe des Objektes, mit *C_GetAttributeValue* der Wert eines Attributes ermitteln. Mit *C_SetAttributeValue* lässt sich ein Wert eines Attributes verändern.
Auf einem Token kann man nach Objekten suchen, indem man *C_FindObjectsInit* aufruft. Mit *C_FindObjects* kann man sich die gefundenen Objekte übergeben lassen und mit *C_FindObjectsFinal* beendet man die Suche. [28]

Funktionen zum ver- und entschlüsseln und Message Digesting

Die Funktionskategorie "Encryption functions" und die der "Decryption functions" enthalten alle Funktionen die Cryptoki zum Ver- oder Entschlüsseln zur Verfügung stellt. In der Kategorie "Message digesting functions" sind Funktionen, die zum Hashen benutzt werden.
Abbildung 3 stellt einen typischen Ablauf der Funktionen dar -siehe Tabelle 7.

C_EncryptInit	*C_Encrypt*	*C_EncryptUpdate*
C_EncryptFinal	*C_DecryptInit*	*C_Decrypt*
C_DecryptUpdate	*C_DecryptFinal*	*C_DigestInit*
C_Digest	*C_DigestUpdate*	*C_DigestKey*
C_DigestFinal		

Tabelle 7: *PKCS#11: Funktionen zum Ver- und Entschlüsseln sowie Message Digest*

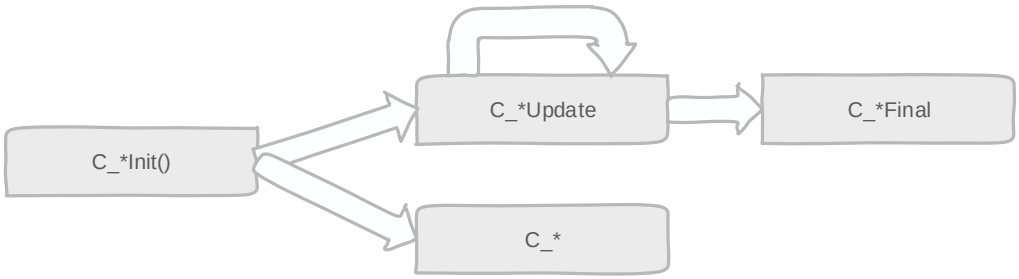

Abbildung 3: *Cryptoki Ablaufschema kryptographischer Funktionen*

Mit der C_*Init, wobei "*"*Encrypt*, *Decrypt* oder *Digest* sein kann, bereitet man die entsprechende kryptographische Funktion vor. Dann wird entweder einmal z.B. *C_Encrypt* aufgerufen oder wiederholt *C_EncryptUpdate*. Mit einem *C_EncryptFinal* wird der Vorgang beendet. Über C_*Init wird angegeben, welcher Mechanismus verwendet wird. Abhängig davon sind weitere Argumente zu übergeben.
C_* nimmt die zu verarbeiteten Daten entgegen und liefert das Ergebnis. Bei größeren Datenmengen, wenn der entgegennehmende Buffer nicht groß genug ist oder die Daten aus einem anderen Grund nicht in einem Schritt verarbeitet werden sollen, wird $C\_*Update$ verwendet. C_*Final muss nur aufgerufen werden, wenn $C\_*Update$ oder *C_DigestKey* verwendet wurde. *C_DigestKey* nimmt ein key-Objekt entgegen und hasht dieses. [28]

Funktionen zum Signieren und Verifizieren

C_SignInit	*C_Sign*	*C_SignUpdate*
C_SignFinal	*C_SignRecoverInit*	*C_SignRecover*
C_VerifyInit	*C_Verify*	*C_VerifyUpdate*
C_VerifyFinal	*C_VerifyRecoverInit*	*C_VerifyRecover*

Tabelle 8: *PKCS#11: Funktionen zum Signieren und Verifizieren*

In den beiden Kategorien “Signing and MACing functions” und “Functions for verifying signatures and MACs”, siehe Tabelle 8, stellt Cryptoki Funktionen bereit, um Daten zu signieren und Signaturen zu verifizieren. Die beiden Kategorien wurden in

dem neuen Entwurf zusammengelegt [27].
Es gibt zwei unterschiedliche Vorgänge: *C_Sign* für einen Signiervorgang und *C_SignRecover*, um mit der Signatur im Anschluss Daten wiederherstellen zu können.
Die Reihenfolge der Funktionsaufrufe ist analog wie bei den vorherigen Kategorien. Mit den *C_Verify* im Namen enthaltenen Funktionen lassen sich Signaturen überprüfen und aus ihnen die Daten wiederherstellen. Mit den Funktionen, die *C_Sign* im Namen enthalten, lassen sich Daten signieren. [28]

Mehrzweckfunktionen

C_DigestEncryptUpdate	*C_DecryptDigestUpdate*	*C_SignEncryptUpdate*
C_DecryptVerifyUpdate		

Tabelle 9: *PKCS#11: Mehrzweckfunktionen*

Die "dual-function cryptographic functions" , siehe Tabelle 9, sind vier Funktionen, die verschiedene, bereits vorgestellte Funktionen kombinieren, um einen möglichst geringen Datenverkehr mit dem Token zu haben.
Zum Verwenden dieser Funktionen müssen vorher die entsprechenden *C_*Init* Funktionen aus den Kategorien aufgerufen worden sein. [28]

Schlüsselverwaltung

C_GenerateKey	*C_GenerateKeyPair*	*C_WrapKey*
C_UnwrapKey	*C_DeriveKey*	

Tabelle 10: *PKCS#11: Schlüsselverwaltung*

Die "key management functions", siehe Tabelle 10, bieten Funktionen, um Schlüssel zu generieren, zu wrappen oder neue Schlüssel durch einen existierenden zu erzeugen. Welche Schlüssel mit *C_GenerateKey* und *C_GenerateKeyPair* erzeugt werden, wird durch den angegeben Mechanismus und einenem vorher erzeugtem Template festgelegt. *C_DeriveKey* erzeugt aus einem bereits existierenden Schlüssel einen Neuen. Dabei wird auch ein Mechanismus angegeben, welcher festlegt, mit welchem Verfahren der neue Schlüssel vom bereits existierenden abgeleitet wird.
Die unterstützte Länge muss vorher vom Token durch *C_GetMechanismInfo* in Erfahrung gebracht werden. Mit dem Erzeugen eines Schlüssels oder eines Schlüsselpaares wird ein entsprechendes key-Objekt erstellt.
Mögliche definierte Schlüsseltypen sind unter anderem: RSA, Blowfish und AES. [28]

Zufallszahlen erzeugen

Um Zufallszahlen oder Pseudozufallszahlen zu erzeugen, bietet Cryptoki zwei Funktionen -siehe Tabelle 11. *C_GenerateRandom* generiert Zufallszahlen. Mit *C_SeedRandom*

C_SeedRandom	*C_GenerateRandom*

Tabelle 11: *PKCS#11: Zufallszahlen erzeugen*

kann dem Zufallsgenerator ein zusätzlicher Seed hinzugefügt werden. Eine allgemein gültige Aussage über die Qualität der Zufallszahlen lässt sich nicht treffen, da die Implementation modul- bzw. tokenspezifisch ist. [28]

Parallele Ausführung

C_GetFunctionStatus	*C_CancelFunction*

Tabelle 12: *PKCS#11 Funktionskategorie: Parallel function management functions*

Die Funktionen der "Parallel function management functions", siehe Tabelle 12, sind im Standard nur noch aus Abwärtskompatibilitätsgründen enthalten. Der Rückgabewert aller dieser Funktionen sollte *CKR_FUNCTION_NOT_PARALLEL* sein. [28]

Callback Funktionen

Cryptoki ermöglicht es, einer Session Funktionen zu übergeben, die bei bestimmten Ereignissen aufgerufen werden. Die Funktionskategorie an sich enthält keine Funktionen. Eine im Standard explizit erwähnte Variante von callback Funktionen wären die "surrender callbacks". Dieser Begriff beschreibt, dass Cryptoki bei langen Operationen dem Programm die Kontrolle zurück gibt. Ein aufgeführtes Beispiel wäre, dass es so möglich ist, dem Nutzer in regelmäßigen Abständen zu signalisieren, dass das Programm noch arbeitet.
Wenn der callback unbekannt ist, dann soll im Zweifel in der callback Funktion direkt CKR_OK zurückgegeben werden. Mit dem Ende der callback Funktion wird der aufrufenden Funktion die Kontrolle zurückgegeben. Herstellereigene callback Funktionen sind möglich. [28]

3 Verwandte Arbeiten

In diesem Kapitel werden kurz verschiedene Projekte vorgestellt, die sich mit dem Testen von kryptographischer Hardware beschäftigen.

3.1 Mozilla Testsuite

Zum Feststellen, ob ein bestimmtes Token mit der Netscape Software von Mozilla funktioniert, kann man laut eine entsprechende Test Suite verwenden [19] [18]. Die verwendeten Programme befinden sich im Webarchiv von Mozilla. Eine Suche außerhalb dieser Archive bringt einen Draft hervor [34]. Nach diesem funktioniert ein aktuelles Testtool nur mit den Mozilla eigenen Network Security Services [20].

3.2 p11-tests

p11-tests ist ein Tool, um ein Token im Rahmen von PKCS #11 zu testen ist [26]. Die letzte Änderung ist aus dem Jahre 2012. Nach [29] lassen sich mit der Nutzung dieses Tools ähnliche Informationen erschließen wie es Ziel dieser Arbeit ist.

3.3 OpenSCDP

OpenSCDP steht für Open Smart Card Development Plaftform und ist eine Sammlung von Tools die zum Entwickeln, Testen und Veröffentlichen von Smartcard Applikationen geeignet ist [22]. Diese müssen in ECMAScript[1] geschrieben sein. Die Tools von OpenSCDP sind in Java geschrieben.
Das Projekt stellt bereits einige Skripte zum Testen verschiedener Anwendungsfälle zur Verfügung [25]

3.4 OpenSC

Das OpenSC Projekt ist ein crossplatform kompatibles Open Source Projekt. Veröffentlicht wird es zurzeit auf Github [23] unter der " LGPL license version 2.1 ". Das Projekt

[1] Der standardisierte Sprachkern von JavaScript

umfasst Tools und Bibliotheken, um mit Smartcards zu arbeiten, wie auch eine PKCS #11 Middleware. Diese ermöglicht unterstützte Smartcard über PKCS #11 zu verwenden. Das Hauptaugenmerk liegt auf auf Smartcards mit Kryptographischen Funktionen und deren Verwendung zu vereinfachen [24].

4 Entwurf

In Kapitel 2 wurde erklärt, was eine kryptographische Hardware ist und wie diese beschaffen sein kann. Dazu wurden einige Projekte in Kapitel 3 vorgestellt, die sich mit kryptographischer Hardware beschäftigen.
Alle Aspekte kryptographischer Hardware zu testen, würde den Rahmen dieser Arbeit übersteigen. Physikalische Tests wie solche auf Wasserdichtheit, Stoßfestigkeit liegen nicht im Bereich der Möglichkeiten dieser Arbeit.
Aus Gründen des Budgets und des Zeitrahmens, kann diese Arbeit nicht alle Kombinationen physikalischer und logischer Schnittstellen von kryptographischer Hardware abdecken.
Der Umfang wird sich auf einige Tests mit Cryptoki begrenzen. Dies bedeutet, dass der Entwurf und die Implementation sich auf jene Hardware beschränken wird, die Cryptoki unterstützt.
In Kapitel 3 wurden Projekte vorgestellt, mit denen unter anderem auf ein Token zugegriffen werden kann. Beispielsweise enthält das OpenSC Projekt (Abschnitt 3.4) ein Tool, das Metainformationen wie den Hersteller und unterstützte Mechanismen eines Cryptoki unterstützenden Tokens auslesen kann.
Ziel dieses Kapitels ist es, Testfälle zu erarbeiten und vorzustellen, die Informationen ermitteln, die über die reine Auslese des Gerätes hinausgehen.
Dafür werden die Funktionen von Cryptoki in Hinsicht auf mögliche Anwendungs- und dadurch Testfälle genauer analysiert. Ein entsprechender Entwurf einer Software wird im Anschluss vorgestellt. Der Name der Software ist TCH.

4.1 Anforderungen

4.1.1 Anwendungsgebiete

Die Software soll dem Anwender ermöglichen, genauere Informationen über ein Token zu erlangen. Dabei wird die Kenntnis um die Existenz verschiedener kryptographischer Funktionen vorausgesetzt.
Ein Entwickler soll anhand der Ausgabe eine Entscheidungsgrundlage erhalten, mit welcher er das richtige Token für seine Zwecke auswählen kann.
Ein Nutzer kryptographischer Hardware soll Informationen über die ihm vorliegende Hardware bekommen.
Es muss möglich sein, dass die Software zumindest in eingeschränkter Funktionsweise

auch bei Hardware, die sich bereits in Verwendung befindet, also konfiguriert ist, genutzt werden kann.

Die folgenden zwei Abschnitte stellen diese zwei typischen Anwendungsfälle dar.

Nutzer: Entwickler

Name	Möglichkeiten testen
Kurzbeschreibung	Ein Entwickler möchte herausfinden, ob das ausgewählte Token sich für seinen Anwendungsfall eignet.
Akteure	Entwickler
Vorbedingung	Der Nutzer hat eine erweiterte Kenntnis von der Materie und weiß, auf was er achten muss. Außerdem befinden sich auf dem Token keine Daten, die nicht verloren gehen dürfen.
Ergebnis	Dem Nutzer liegen die Testergebnisse des Tokens vor.
Nachbedingung	Das Token sollte nicht zerstört sein.
Ablauf	Der Nutzer startet die Software und gibt notwendige Daten als Argument an. Dann arbeitet die Software. Eventuell ist eine Nutzerinteraktion notwendig.

Tabelle 13: *Anwendungsfall: Möglichkeiten testen*

Wie in Kapitel 1 bereits aufgeführt, ist das Ziel dieser Arbeit, kryptographische Hardware zu testen und dem Anwender anhand der Ergebnisse dieser Tests, eine Entscheidungsgrundlage zu geben. Tabelle 13 führt den Anwendungsfall *Möglichkeitentesten* auf.
Die Situation könnte sein, dass ein Entwickler ein System entwickelt, bei dem kryptographische Hardware zum Einsatz kommen soll.
Ein mögliches Szenario wäre, dass jedem Nutzer ein Token mit einem persönlichen Schlüssel ausgehändigt werden soll, mit dem dieser sich vor bestimmten Systemen ausweisen kann. Es wird vorausgesetzt, dass der Entwickler -in diesem Falle der Nutzer von TCH- über einen hohen Kenntnisstand der Materie verfügt. Das bedeutet mindestens, dass ihm der Inhalt des PKCS #11 Standards [28] geläufig ist. Dieser Entwickler kann mit TCH und einer Auswahl an entsprechenden Tokens nun herausfinden, welches sich für seine Zwecke eignet.

Nutzer: Anwender

Tabelle 14 stellt einen weiteren möglichen Anwendungsfall namens *WeitereAnwendungsgebietefinden* dar. Der bedeutende Unterschied zum Vorherigen ist, dass der Anwender nun ein bereits konfiguriertes Token besitzt. Es soll also keine Auswahl getroffen, sondern herausgefunden werden, was mit dem Token in der aktuellen Konfiguration möglich ist und wie diese aussieht. Dabei darf die Konfiguration nicht modifiziert werden, es sei denn, der Nutzer erlaubt dies explizit.

Name	Weitere Anwendungsgebiete finden
Kurzbeschreibung	Ein Nutzer hat ein konfiguriertes Token erhalten und möchte herausfinden, was er mit dem Token machen kann, ohne die Konfiguration zu verändern.
Akteure	Tokenbesitzer
Vorbedingung	Der Nutzer weiß nicht, wie die aktuelle Konfiguration ist. Auf dem Token befinden sich Daten, die nicht verloren gehen dürfen.
Ergebnis	Dem Nutzer liegen die Testergebnisse des Tokens vor, durch die deutlich wird, wozu er das Token noch nutzen kann.
Nachbedingung	Die Konfiguration wurde nicht verändert.
Ablauf	Der Nutzer startet die Software und gibt notwendige Nutzerdaten als Argument an. Dann arbeitet die Software. Eventuell ist eine Nutzerinteraktion notwendig.

Tabelle 14: *Anwendungsfall: Weitere Anwendungsgebiete finden*

4.2 Analyse

4.2.1 Abhängigkeitsanalyse

Im folgenden Abschnitt werden die Abhängigkeiten von den Cryptoki Funktionen untereinander genauer untersucht, um so mögliche Testfälle zu erkennen.
Da TCH mit verschiedenen, vorher unbekannten Token funktionieren soll, muss das PKCS #11 Modul von TCH geladen werden. Wie bereits in Kapitel 2 erwähnt, können dann mit der Funktion *C_GetFunctionList* die Zeiger auf die jeweiligen Funktionen erhalten werden. Es werden beim Entwurf so wenig Einschränkungen wie möglich gemacht, da die Anwendungsfälle sich aus den Möglichkeiten ergeben.

Einige der Funktionen von PKCS #11 stehen in direkten Abhängigkeiten, so kann ohne *C_Initialize* kaum eine weitere Funktion erfolgreich, also mit dem Rückgabewert *CKR_OK*, ausgeführt werden. Der entsprechende Rückgabewert von Funktionen, die trotzdem aufgerufen werden, sollte *CKR_CRYPTOKI_NOT_INITIALIZED* sein.
Andere Funktionen besitzen keine direkten Abhängigkeiten. Zum erfolgreichen Ausführen dieser werden aber Daten benötigt, die bei der Ausführung einer anderen Funktion entstehen. So braucht man nicht explizit mit *C_GetSlotList* eine Slot Nummer zu ermitteln. Da diese aber nicht von null aufwärts nummeriert sein müssen, lässt sich eine Slot Nummer eines unbekannten Tokens nur erraten. Im weiteren Verlauf der Arbeit wird davon ausgegangen, dass alle Informationen, die über das Token vorhanden sind, durch vorherige Ausführung der entsprechenden Funktionen erlangt wurden.
Tabelle 15 unterteilt die Funktionen von PKCS #11 in vier Abhängigkeitsgruppen. Die Unterteilung geschieht nach Abhängigkeiten die über die Funktionsgruppengrenze hinausgeht. Die erste Abhängigkeitsgruppe mit der Ziffer 0 drückt aus, dass die Funktionen dieser Gruppe losgelöst von Sessions, Token IDs und einem eingeloggten

Abhängigkeitsgruppe	Funktionen
0	*C_Initialize*, *C_Finalize*, *C_GetInfo*, *C_GetFunctionList*, *C_GetSlotList*
1	*C_GetSlotInfo*, *C_GetTokenInfo*, *C_WaitForSlotEvent*, *C_GetMechanismList*, *C_GetMechanismInfo*, *C_InitToken*, *C_OpenSession*, *C_CloseAllSessions*
2	*C_CloseSession*, *C_GetSessionInfo*, *C_GetOperationState*, *C_SetOperationState*, *C_Login*, *C_Logout*, *C_CreateObject*, *C_CopyObject*, *C_DestroyObject*, *C_GetObjectSize*, *C_GetAttributeValue*, *C_SetAttributeValue*, *C_FindObjectsInit*, *C_FintObjects*, *C_FindObjects*, *C_EncryptInit*, *C_Encrypt*, *C_EncryptUpdate*, *C_EncryptFinal*, *C_DecryptInit*, *C_Decrypt*, *C_DecryptUpdate*, *C_DecryptFinal*, *C_DigestInit*, *C_Digest*, *C_DigestUpdate*, *C_DigestKey*, *C_DigestFinal*, *C_SignInit*, *C_Sign*, *C_SignUpdate*, *C_SignFinal*, *C_SignRecoverInit*, *C_SignRecover*, *C_VerifyInit*, *C_Verify*, *C_VerifyUpdate*, *C_VerifyFinal*, *C_VerifyRecoverInit*, *C_VerifyRecover*, *C_DigestEncryptUpdate*, *C_DecryptDigestUpdate*, *C_SignEncryptUpdate*, *C_DecryptVerifyUpdate*, *C_GenerateKey*, *C_GenerateKeyPair*, *C_WrapKey*, *C_UnwrapKey*, *C_DeriveKey*, *C_SeedRandom*, *C_GenerateRandom*
3	*C_InitPIN*, *C_SetPIN*

Tabelle 15: *Unterteilung der PKCS #11 Funktionen in minimale Abhängigkeitsgruppen*

User erfolgreich ausgeführt werden können. Maximale Abhängigkeit von Funktionen in dieser Gruppe darf die von *C_Initialize* sein. Die zweite Abhängigkeitsgruppe fasst die Funktionen zusammen, die eine Slot bzw. Token ID verwenden. Die Dritte, die Funktionen, die eine Session benötigen und die vierte Abhängigkeitsgruppe die, die einen Login brauchen.
Die Tabelle zeigt nicht auf, welche weiteren Abhängigkeiten unter den Funktionen bestehen oder ob es einen Unterschied macht, ob man die Funktion mit dem User, dem SO oder gar nicht eingeloggt aufruft. Die Tabelle zeigt lediglich an, was das minimal notwendige ist, um eine Funktion erfolgreich auszuführen.
Ziel der folgenden Absätze ist es die weiteren Abhängigkeiten zu klären, um so die möglichen Anwendungsfälle indirekt zu beschreiben.
Dabei werden die Funktionen wie in Kapitel 2.3.2 gruppiert. Die Gruppen “Parallele Ausführung“ und “Callback Funktionen“ sind nicht aufgeführt. Die Funktionen der ersten Gruppe werden nicht mehr unterstützt und die “Callback Funktionen” sind eigens definierte Funktionen. Die Abhängigkeiten sind also anwendungsspezifisch. Lediglich eine Session wird vorausgesetzt.

Allgemeine Funktionen

C_Initialize und *C_GetFunctionList* sind unabhängig von anderen Funktionen. *C_Finalize* und *C_GetInfo* hingegen können nur erfolgreich ausgeführt werden, wenn *C_Initialize* erfolgreich ausgeführt wurde. Dies wurde schon in Tabelle 15 deutlich.

Slot und Token Verwaltung

Wie in der Tabelle 15 ersichtlich, benötigen alle Funktionen dieser Gruppe (außer *C_GetSlotList*) ein erfolgreiches Ausführen von *C_GetSlotList*. Genauer wird hier eine Slot bzw. Token Nummer gebraucht.
C_InitPIN und *C_SetPIN* benötigen eine gültige Session. *C_InitPIN* kann nur erfolgreich ausgeführt werden, wenn die Session eine solche mit Lese- und Schreibberechtigung ist und man darüber hinaus als SO angemeldet ist. Also ist *C_InitPIN* weiterhin von *C_Login* abhängig.

Session Funktionen

C_OpenSession und *C_CloseAllSessions* benötigen eine Slot Nummer. *C_CloseSession*, *C_GetSessionInfo*, *C_GetOperationState*, *C_SetOperationState*, *C_Login* und *C_Logout* benötigen eine Session, die durch *C_OpenSession* generiert werden kann. *C_Logout* kann zwar ausgeführt werden, ohne vorher einen Login auszuführen, gibt dann aber den Fehlerwert *CKR_USER_NOT_LOGGED_IN* zurück.
C_SetOperationState benötigt für eine erfolgreiche Ausführung Werte, die mittels *C_GetOperationState* ermittelt wurden.

Objektverwaltung

Alle Funktionen dieser Gruppe benötigen eine Session. Die Funktionen zum Handeln eines existierenden Objektes also *C_GetObjectSize*, *C_CopyObject*, *C_GetAttributeValue*, *C_DestroyObject*, *C_SetAttributeValue* benötigen einen Verweis auf ein Objekt. Sie sind also abhängig von *C_CreateObject*, *C_FindObjects* oder einer anderen Funktion, die Objekte erzeugt, wie die der Schlüsselerzeugung.
Das Programm hat immer nur Zugriff auf Dateien mit entsprechenden Zugriffsbeschränkungen. Mit einem Login des Users hat man also Zugriff auf dessen Dateien. Ein Login ist jedoch nicht zwingend erforderlich, um auf Dateien zuzugreifen. Dies wurde bereits in Abschnitt 2.3.2 erläutert.

Funktionen zum ver- und entschlüsseln und Message Digesting & Funktionen zum Signieren und Verifizieren

Alle Funktionen dieser Gruppe benötigen eine Session. *C_EncryptInit* und *C_DecryptInit* benötigen einen Verweis auf den zu verwendenden Schlüssel. Dieser kann entweder über *C_FindObjects* oder eine Funktion zum Erzeugen des entsprechenden Schlüssels erreicht werden (*C_GenerateKey*, *C_GenerateKeyPair*).
Weiterhin gilt, dass die Funktionen dieser Gruppen nur in dem bereits in den Grundlagen 2.3.2 und 2.3.2 vorgestellten Ablaufschema richtig funktionieren.

Mehrzweckfunktionen

Die Mehrzweckfunktionen bringen keine neue Funktionalität ein. Sie sind eine Bündelung der bereits existierenden Funktionen. Es gelten die Abhängigkeiten der Funktionen die hier kombiniert werden.

Schlüsselverwaltung

Wie in Tabelle 15 deutlich wird, benötigen alle Funktionen dieser Gruppe eine Session. Weiterhin greifen die Funktionen auf Objekte zu bzw. erstellen welche. Bei den Funktionen *C_WrapKey* und *C_DeriveKey* ist also minimal notwendig, dass das entsprechende Zugriffsrecht besteht und die Objekte gefunden werden können. Bei einem privaten Schlüssel, der auf dem Token gespeichert ist, wird also eine gültige Session mit einem entsprechenden Login benötigt.

Zufallszahlen erzeugen

Die Funktionen dieser Gruppe haben keine weiteren Abhängigkeiten als die in der Tabelle 15 dargestellten.

4.3 Ablauf

Nachdem nun die Abhängigkeiten dargestellt wurden, wird in diesem Abschnitt der darauf basierende Entwurf von TCH vorgestellt.
Die Funktionsweise von TCH lässt sich grob in drei Schritte gliedern.

1. Funktionen auf Implementation testen
2. Informationen auslesen
3. Mechanismen und Funktionen durch Testfälle testen

Im ersten Schritt werden alle Funktionen des Moduls einmal aufgerufen, was eine Übersicht über die implementierten Funktionen gibt. Sind die entsprechenden Funktionen implementiert, können die Informationen des Tokens ausgelesen werden. Dazu gehören Metainformationen wie die möglichen Mechanismen, aber auch die auf dem Token

gespeicherten Objekte. Im letzten Schritt werden die Funktionen und deren Anwendungsfälle durch Testfälle getestet. Da es sehr viele Anwendungsfälle gibt, werden die Testfälle so allgemein wie möglich gehalten.
In den drei folgenden Abschnitten werden die eben genannten drei Schritte genauer erläutert.

4.3.1 Funktionen auf Implementation testen

In einer Cryptoki Implementation müssen nicht alle Funktionen des Standards implementiert sein [28]. Wenn eine Funktion nicht unterstützt wird, sollte in der Funktionszeigerliste, die durch *C_GetFunctionList* ermittelt wird, diese Funktion dennoch enthalten sein. Der Rückgabewert dieser Funktion sollte dann *CKR_FUNCTION_NOT_SUPPORTED* sein. Ist der Rückgabewert ein anderer, so sollte sich erschließen lassen, dass die Funktion unterstützt wird. Inwieweit die Funktion ein erwartetes Ergebnis in gemessener Zeit berechnet, wird in diesem Schritt noch nicht getestet. Dieser Schritt soll einen Anhaltspunkt für das Testen der Funktionen in Schritt drei darstellen. Wird eine Funktion nicht unterstützt, können unter Umständen andere Funktionen auch nicht mehr erfolgreich ausgeführt werden und somit können sie auch nicht getestet werden. Entsprechende Abhängigkeiten wurden in Kapitel 4.2.1 aufgezeigt.
Das Resultat dieses Schrittes ist eine Liste von Funktionen und der Information, ob diese unterstützt werden.

4.3.2 Informationen auslesen

Metainformationen auslesen

Wenn die entsprechenden Funktionen nach Schritt eins unterstützt werden, dann können Metainformationen ausgelesen werden. Welche und wie diese ermittelt werden, ist in Tabelle 16 ersichtlich. Es sollte sich durch den Aufruf von *C_GetMechanismList* und die Iteration über alle Mechanismen und die Verwendung von *C_GetMechanismInfo* die gleiche Liste ergeben. Dies ist aber ein Testfall und wird nicht in diesem Schritt überprüft. In diesem Schritt wird noch nicht festgestellt, ob ein Mechanismus tatsächlich implementiert wurde. Dies ist erst durch das Ausführen der Funktionen mit dem entsprechenden Mechanismus festzustellen.

Objekte auslesen

Über die Suchfunktionen in der Objektverwaltungsgruppe lassen sich Objekte finden. Die Suchfunktionen nehmen ein Template entgegen. Dieses Template ist ein Muster von dem Objekt, siehe Abschnitt 2.3.2, nach dem man sucht. Das kann z.B. ein Labelinhalt sein.
Ziel dieses Schrittes ist es, einen Überblick darüber zu geben, welche Objekte sich auf

C_GetInfo	Füllt eine entsprechende Struktur mit allgemeinen Informationen über die Cryptoki Implementation. Genauer wird die Versionsnummer, der Hersteller, ein Flagfeld, ein Feld für die Beschreibung der Implementation und die eigene Implementationsversionsnummer zurückgegeben.
C_GetSlotInfo	Füllt eine entsprechende Struktur mit Informationen über einen Slot. Die Informationen sind eine Beschreibung, eine Herstellernummer, ein Flagfeld, ein Feld mit der Hardwareversionsnummer und eines mit der Firmwareversionsnummer. Es gibt drei unterschiedliche Flags, die gesetzt sein können. Diese kennzeichnen, ob ein Token in dem Slot ist, ob das Lesegerät ein Entfernen der Token ermöglicht und ob es sich um einen Hardwaretoken handelt.
C_GetTokenInfo	Füllt eine entsprechende Struktur mit Informationen über ein Token. Darunter befinden sich die Informationen wie die möglichen PIN-Längen, den freien Speicher und weitere Informationen, die das Token beschreiben.
C_GetMechanismList	Füllt eine entsprechende Struktur mit einer Liste der unterstützten Mechanismen zu einem Token.
C_GetMechanismInfo	Füllt eine entsprechende Struktur mit Informationen über einen Mechanismus, der von dem Token unterstütz wird. In dieser Struktur ist die minimale sowie die maximale Schlüssellänge und ein Flagfeld enthalten. Das Flagfeld gibt Aufschluss darüber, welche Funktionen mit diesen Mechanismen durchgeführt werden können und ob der Mechanismus auf der Hardware durchgeführt wird.

Tabelle 16: *Quelle von Metainformationen über das Token*

dem Token befinden. Ziel ist es nicht, alle möglichen Informationen über Schlüssel anzuzeigen oder zu testen, ob sich diese extrahieren lassen.

4.3.3 Mechanismen und Funktionen durch Testfälle testen

Mit den Abhängigkeiten, die in Abschnitt 4.2.1 beschrieben wurden, ergeben sich die möglichen Anwendungsfälle von Cryptoki.
Im folgenden Abschnitt werden nun einige Testfälle vorgestellt, die sich daraus ergeben. In den einzelnen Abschnitten werden nur die zu testenden Funktionen genannt. Die weiteren dafür verwendeten Funktionen ergeben sich aus den Abhängigkeiten in Abschnitt 4.2.1. Ein Testfall wird nur dann ausgeführt, wenn alle nicht optionalen Funktionen implementiert sind. Optionale Funktionen sind mit einem * gekennzeichnet. Eine Funktion zu testen bedeutet im Folgenden, zu versuchen, eine Funktion erfolgreich auszuführen

und, wenn möglich, das Ergebnis zu überprüfen.
Bei Testfällen die mit Mechanismen agieren, wird wenn möglich über alle unterstützten Mechanismen iteriert. Da bei Funktionsaufrufen bei verschiedenen Mechanismen auch unterschiedliche Parameter erwartet werden, ist es nicht möglich, alle Mechanismen gleich zu testen. Da die Mechanismen auch völlig unterschiedliche Algorithmen repräsentieren, ist dies auch nur bedingt sinnvoll.
Die Funktion *C_GetFunctionList* wird in keinem Testfall gesondert geprüft. Sollte diese Funktion nicht wie erwartet funktionieren, dann ist ein Starten der Tests nicht möglich, da die Funktionszeiger fehlen. Daraus ließe sich schließen, dass ein Verwenden des Tokens mit einer Software, die über den Mechanismus, der von Cryptoki extra dafür zur Verfügung gestellt wird, nicht möglich ist.

Testfall 1: Minimal

Zu testende Funktionen: *C_Initialize*, *C_GetInfo* und *C_Finalize*.
Es soll geprüft werden, ob das Initialisieren von Cryptoki funktioniert und eine Funktion nach Initialisierung erfolgreich ausgeführt werden kann. Um sicherzugehen, dass Funktionen nicht ohne Initialisierung ausgeführt werden können, wird *C_Finalize* zu Beginn einmal ausgeführt und der Rückgabewert überprüft.

Testfall 2: SlotInformation

Zu testende Funktionen: *C_GetSlotList* und *C_GetSlotInfo*
In diesem Testfall werden die slotbezogenen Funktionen getestet. Erst wird *C_GetSlotList* dann *C_GetSlotInfo* ausgeführt. Mit *C_GetSlotInfo* wird das Ergebnis von *C_GetSlotList* direkt überprüft.

Testfall 3: TokenInformation

Zu testende Funktionen: *C_GetSlotList*, *C_GetTokenInfo* und *C_GetMechanismList**.
In diesem Testfall werden die tokenbezogenen Funktionen getestet. *C_GetSlotList* wird in diesem Testfall im Gegensatz zum vorher genannten verwendet, um aktive Tokens zu finden. Mit diesen wird dann *C_GetTokenInfo* und *C_GetMechanismList* ausgeführt.

Testfall 4: CompareMechanismLists

Zu testende Funktionen: *C_GetMechanismList* und *C_GetMechanismInfo*
Ziel dieses Testfalles ist es, zu überprüfen, ob die Liste, die über *C_GetMechanismList* übergeben wird, gleich der Liste ist, die sich über das Iterieren mittels *C_GetMechanismInfo* von allen bekannten Mechanismen ergibt. Dazu wird mittels *C_GetMechanismList* eine Liste erzeugt. Dann wird über alle definierten Mechanismen iteriert und *C_GetMechanismInfo* ausgeführt. Dabei entsteht eine zweite Liste. Diese beiden werden dann verglichen.

Testfall 5: SlotEvents

Zu testende Funktionen: *C_WaitForSlotEvent* und *C_GetSlotInfo*
In diesem Testfall wird getestet, ob das Entfernen und Hinzufügen des Tokens im Slot richtig erkannt wird. Dabei wird von einem bereits erkannten Token im Slot ausgegangen. Dies wird über die Flags in der Struktur, die über *C_GetSlotInfo* ermittelt wird, geprüft. Dann wird *C_WaitForSlotEvent* aufgerufen. Die Funktion blockiert das Programm, bis ein Event auftritt. Tritt ein Event auf, wird wieder mit *C_GetSlotInfo* überprüft, ob das Fehlen des Tokens richtig erkannt wird. Dann wird wieder *C_WaitForSlotEvent* aufgerufen und auf ein Event gewartet. Dies sollte das Einstecken des Tokens sein. Geprüft wird dies wieder über *C_GetSlotInfo*.

Testfall 6: SingleSessionWithUserLogin

Zu testende Funktionen: *C_OpenSession*, *C_GetSessionInfo*, *C_CloseSession*, *C_Login* und *C_Logout*.
Dieser Testfall testet den Umgang des Tokens mit einer einzelnen Session. Nach jedem sessionverändernden Funktionsaufruf wird *C_GetSessionInfo* ausgeführt, um die Änderung zu verifizieren. Dabei werden alle zulässigen Kombinationen der Sessionflags durchgegangen.

Testfall 7: MultipleSessionWithUserLogin

Zu testende Funktionen: *C_OpenSession*, *C_GetSessionInfo*, *C_CloseAllSession*, *C_Login* und *C_Logout*.
Dieser Testfall ähnelt dem Vorherigen. Unterschiedlich ist hier, dass getestet wird, ob der Umgang mit mehreren Sessions korrekt ist. Dabei wird insbesondere getestet, ob der Loginstatus bei Änderung auf die anderen Sessions übernommen wird. Auch hier werden alle zulässigen Kombinationen der Sessionflags durchgegangen.

Testfall 8: PreserveOperationState

Zu testende Funktionen: *C_GetOperationState* und *C_SetOperationState*.
In diesem Testfall wird eine einfache Session geöffnet und überprüft, ob sich diese nach dem Ausführen von *C_GetOperationState* und anschließendem Schließen wieder über *C_SetOperationState* herstellen lässt. Der Vorgang wird bei einem erweiterten Digestingvorgang wiederholt.

Testfall 9: AdminToken

Zu testende Funktionen: *C_InitToken* und *C_InitPIN*.
In diesem Testfall soll getestet werden, ob sich das Token administrieren lässt. Dazu wird das Token, wenn möglich, neu initialisiert und anschließend die User PIN initialisiert.

Testfall 10: ChangePin

Zu testende Funktionen: *C_SetPIN*.
In diesem Testfall wird einmal für den Security Officer und einmal für den User versucht, die PIN zu ändern. Dabei wird nach erfolgreichem Ausführen von *C_SetPIN* ein Einloggen mit der neuen PIN versucht.

Testfall 11: ObjectHandling

Zu testende Funktionen: *C_CreateObject*, *C_GetAttributeValue**, *C_SetAttributeValue**, *C_GetObjectSize**, *C_CopyObject** und *C_DestroyObject*.
In diesem Testfall wird der Umgang mit Objekten getestet. Dabei wird ein Großteil der objektbezogenen Funktionen ausgeführt. Jede Objektart, also key, certificate und data, wird einmal erzeugt, kopiert, manipuliert und zerstört.

Testfall 12: ObjectSearch

Zu testende Funktionen: *C_FindObjectsInit*, *C_FindObjects* und *C_FindObjectsFinal*.
In diesem Testfall wird die Objektsuche getestet. Es geht nicht darum, wie sich die Suche im Zusammenhang der Anzahl der Objekte verhält, sondern lediglich ob die Funktion, wenn sie als unterstützt ausgewiesen wurde, auszuführen und zu prüfen, ob vorher erstelle Objekte gefunden werden. Durch ein erneutes Starten von TCH kann überprüft werden, ob die Objekte nach dem Schließen der Anwendung bzw. Entfernen des Tokens bestehen bleiben.

Testfall 13: DeAndEncryptionSimple

Zu testende Funktionen: *C_EncryptInit*, *C_Encrypt*, *C_DecryptInit* und *C_Decrypt*.
In diesem Testfall soll das Ver- und Entschlüsseln kleiner Datenmengen getestet werden. Dabei wird mit einem vorher erzeugten key-Objekt eine Datei verschlüsselt. Die anschließend entschlüsselte Datei wird mit dem Ausgangsmaterial verglichen.

Testfall 14: DeAndEncryptionMultiple

Zu testende Funktionen: *C_EncryptInit*, *C_EncryptUpdate*, *C_EncryptFinal*, *C_DecryptInit*, *C_DecryptUpdate* und *C_DecryptFinal*.
Der Ablauf dieses Testfalls ist der gleiche wie bei dem Vorherigen, bis darauf dass anstatt dem einmaligen Aufruf von *C_Encrypt* die Funktionen *C_EncryptUpdate* und *C_EncryptFinal* verwendet werden.

Testfall 15: DigestSimple

Zu testende Funktionen: *C_DigestInit*, *C_Digest* und *C_DigestKey**.
Dieser Testfall prüft, ob das Digesting mit kleinen Datenmengen funktioniert. Dabei

werden die Ursprungsdaten zweimal den Funktionen übergeben, um überprüfen zu können, ob die Ergebnisse gleich sind. So werden grobe Fehler in der Implementierung sichtbar. Dem Nutzer werden das Ausgangsmaterial und die Hashwerte angezeigt, sodass diese überprüft werden können. Zusätzlich wird, wenn *C_DigestKey* unterstützt wird, ein key-Objekt erzeugt und der Funktion *C_DigestKey* übergeben.

Testfall 16: DigestMultiple

Zu testende Funktionen: *C_DigestInit*, *C_DigestUpdate* und *C_DigestFinal*.
Bei diesem Testfall wird im Gegensatz zum Vorherigen anstatt einmalig *C_Digest* durch *C_DigestUpdate*, *C_DigestFinal* ersetzt.

Testfall 17: SignAndVerifySimple

Zu testende Funktionen: *C_SignInit*, *C_Sign*, *C_VerifyInit* und *C_Verify*.
Dieser Testfall soll das Signieren von Daten bzw. das Verifizieren der daraus entstehenden Signaturen testen. Dafür werden Signaturen erzeugt und diese anschließend verifiziert.

Testfall 18: SignAndVerifyMultiple

Zu testende Funktionen: *C_SignInit*, *C_SignUpdate*, *C_SignFinal*, *C_VerifyInit*, *C_VerifyUpdate* und *C_VerifyFinal*,
Bei diesem Testfall werden im Gegensatz zu dem Vorherigen einmalig *C_Sign* bzw. *C_Verify* ausgeführten Funktionen *C_SignUpdate* und *C_SignFinal* bzw. *C_VerifyUpdate* und *C_VerifyFinal* ausgeführt.

Testfall 19: SignAndVerifyRecover

Zu testende Funktionen: *C_SignRecoverInit*, *C_SignRecover*, *C_VerifyRecoverInit* und *C_VerifyRecover*
In diesem Testfall werden der Vorgang des Signierens mit Wiederherstellmöglichkeit und das anschließende Wiederherstellen der Daten durch die so erzeugte Signatur getestet. Also wird erst *C_SignRecoverInit* und *C_SignRecover* ausgeführt und anschließend *C_VerifyRecoverInit* und *C_VerifyRecover*.

Testfall 20: DualPurposeFunctions

Zu testende Funktionen: *C_DigestEncryptUpdate*, *C_DecryptDigestUpdate*, *C_SignEncryptUpdate* und *C_DecryptVerifyUpdate*
In diesem Testfall werden die Mehrzweckfunktionen getestet. Da die Funktionen keine grundsätzlich neuen Funktionalitäten in Cryptoki einbringen, lässt sich dieser Testfall von den jeweiligen expliziten Funktionen ableiten. *C_SignEncryptUpdate*

und *C_DecryptVerifyUpdate* lassen sich zudem miteinander kombinieren, um das Ergebnis des Signier- und Verschlüsselungsvorganges zu verifizieren.

Testfall 21: SingleKeyGeneration

Zu testende Funktionen: *C_GenerateKey*
In diesem Testfall soll das Generieren von key-Objekten getestet werden. Dazu werden die unterstützten Mechanismen des Tokens verwendet und *C_GenerateKey* ausgeführt.

Testfall 22: KeyPairGeneration

Zu testende Funktionen: *C_GenerateKeyPair*
In diesem Testfall wird ebenfalls das Generieren von key-Objekten getestet. Bei diesem Testfall werden -in Abgrenzung zum Vorherigen- Schlüsselpaare erzeugt. Dies geschieht über die Funktion *C_GenerateKeyPair*.

Testfall 23: WrapAndUnwrapKey

Zu testende Funktionen: *C_WrapKey* und *C_UnwrapKey*
Mit der Funktion *C_WrapKey* kann ein privater oder geheimer Schlüssel verpackt werden. Mit *C_UnwrapKey* kann dieser wieder ausgepackt werden. In diesem Testfall wird dieser Vorgang geprüft.

Testfall 24: DeriveKey

Zu testende Funktionen: *C_DeriveKey*
Dieser Testfall prüft, ob das Erzeugen eines Schlüssels aus einem Anderen heraus funktioniert. Wird ein neuer Schlüssel generiert, so kann dieser überprüft werden, indem eine Signierung mit dem Originalschlüssel geprüft wird.

Testfall 25: GenerateRandom

Zu testende Funktionen: *C_SeedRandom** und *C_GenerateRandom*
In diesem Testfall werden die Funktionen getestet, die Zufallszahlen erzeugen. Eine Bewertung dieser Zahlen ist nicht Teil des Testes. Um eine statistische Auswertung zu ermöglichen, werden die ermittelten Werte in einer Datei für den Benutzer von TCH bereitgestellt.

5 Implementierung

In diesem Kapitel ist die Implementierung von TCH dokumentiert.

5.1 Allgemeine Anmerkungen

Dieser Abschnitt führt allgemeine Rahmenbedingungen zur Entwicklung von TCH auf.

5.1.1 Verwendete Tools

TCH ist in C auf und für einen 64bit Prozessor unter Ubuntu 14.04 LTS und den mitgelieferten Hilfsmitteln entwickelt. Da Cryptoki einige Anpassungen für unterschiedliche Betriebssysteme benötigt, ist ein Kompilieren und Ausführen nur unter diesem oder einem ähnlichen Setup ohne weiteres möglich. Das Projekt ist auf Github unter *https://github.com/easymaxi/TCH* veröffentlicht.

5.1.2 Bibliotheken

Zusätzlich zu Cryptoki werden nur Standard C Bibliotheken verwendet. Cryptoki wird in der zu PKCS #11 v2.20 gehörigen Version verwendet.
Mittels *dlopen* und *dlsym* wird die Adresse der *C_GetFunctionList* Funktion ermittelt. Mit dieser werden die Adressen der Funktionen des Modules ermittelt. So können beliebige Module dynamisch zur Laufzeit geladen werden.

5.1.3 Parameter

TCH erkennt sechs verschiedene Parameter.

-o

Mit dem Parameter -o wird angegeben, ob Daten auf dem Token überschrieben bzw. gelöscht werden dürfen. Diese Option sollte nur benutzt werden, wenn das Token nicht produktiv genutzt wird, da es zum Löschen bestehender Objekte auf dem Token kommt.

-i
Mit -i gibt man an, ob das Initialisieren des Tokens getestet werden soll.

-m Datei
Mit -m wird das Modul des zu testenden Tokens übergeben. Ohne diesen Parameter ist TCH nicht effektiv nutzbar.

-u PIN
Über -u wird die PIN des Users übergeben.

-s PIN
Über -s wird die PIN des SOs übergeben.

-v
Über den Parameter -v können erweiterte Ausgaben aktiviert werden. Dies ist insbesondere wichtig, wenn eine Funktion scheitert. Grundsätzlich wird, wenn eine Funktion getestet wird, dieses vorher auf der Konsole ausgegeben und auch das Ergebnis im Anschluss an die Ausführung angezeigt.

Sind PINs über -s oder -u übergeben, wird, wenn möglich immer der Login verwendet. Sollte ein Loginversuch scheitern, wird TCH beendet. So hat der Anwender die Möglichkeit, die Daten zu überprüfen. Dies ist insbesondere deshalb wichtig, da bei wiederholt gescheiterten Anmeldeversuchen der User bzw. SO gesperrt werden kann.

5.2 Implementierung des Ablaufes

In diesem Abschnitt wird die Implementierung des entworfenen Ablaufes aus Abschnitt 4.3 dokumentiert.

5.2.1 Funktionen auf Implementation testen

Um die Funktionen auf Implementation zu testen, reicht es, diese aufzurufen. Die Funktion *testFunctionsImplemented* prüft die Funktionen auf Implementation. Alle Funktionen von Cryptoki werden einmal aufgerufen. Ziel ist es nicht, diese erfolgreich auszuführen, sondern festzustellen, ob sie implementiert sind.
Dafür wird eine Liste mit Strukturen wie in Listing 5.1 dargestellt gefüllt. Jede PKCS #11 Funktion wird durch ein Element dieser Liste repräsentiert. In *funcName* steht der Name der Funktion und wenn die Funktion implementiert, also der Rückgabewert ungleich *CKR_FUNCTION_NOT_SUPPORTED* ist, wird *impl* auf eins gesetzt.

Um sicherzugehen, dass keine unerwünschten Nebeneffekte, wie eine geänderte Konfiguration oder ein fehlerhafter Login der zum Sperren des Users oder SOs führen kann, eintreten, werden grundsätzlich Werte verwendet, die möglichst nicht in einem realen Szenario auftreten können.
Da viele Argumente wie z.B. Objekt- oder Sessionhandles Zahlenwerte sind und sich deren Validität nicht einfach prüfen lässt, ohne die entsprechenden Funktionen korrekt

```
struct FUNCTIONINFO
{
        int impl;
        char *funcName;
};
```

Listing 5.1: *Struktur mit Informationen über eine Funktion*

auszuführen, können Überschneidungen auftreten.
Der Implementationstest in TCH ist so geschrieben, dass, wenn Überschneidungen auftreten, diese trotzdem nicht die Konfiguration des Tokens beeinflussen.
Dabei werden folgende Grundsätze verwendet:

- Wird ein Zeiger erwartet, so übergebe einen Zeiger mit dem Wert *NULL_PTR*.
- Wird ein vordefinierter Zahlenwert erwartet, dann weiche auf einen nicht definierten aus.
- Wird ein Handle auf ein Objekt erwartet, repräsentiert durch eine Zahl, verwende eine beliebige Zahl und stelle sicher, dass der Handle nicht gültig ist.

Ein Ansatz, der nach ersten Tests verworfen wurde, war *C_Initialize* als letztes zu prüfen, damit grundsätzlich das vollständige Ausführen jeglicher weiteren Funktion, außer *C_GetFunctionList*, verhindert wird. Es hat sich herausgestellt, dass das dazu führen kann, dass alle Funktionen als 'nicht unterstützt' ausgewiesen werden. Deshalb wird *C_Initialize* zu Beginn ausgeführt.

5.2.2 Informationen ausgeben

Metainformationen auslesen

Tabelle 16 in Abschnitt 4.3.2 zeigt eine Übersicht der Funktionen, die Aufschluss über die Metainformationen des Tokens geben. Diese werden in TCH nach Vorgaben des Standards pro Token aufgerufen und die Daten ausgegeben. Die Flags werden als ihre entsprechenden Konstantennamen angezeigt. Während dieses Schrittes wird eine Liste der Mechanismen gespeichert, die später von den in Abschnitt 4.3.3 definierten Testfällen genutzt werden kann. Um zu prüfen, ob diese Liste vollständig ist, gibt es den Testfall 4.

Objekte auslesen

Die Objekte auf dem Token werden nach Klassen unterteilt gesucht und die Anzahl angezeigt. Dies ist über die Suchfunktionen der Objektverwaltungsgruppe realisiert. Wird der Parameter -o übergeben, dann werden die gefundenen Objekte zerstört.

5.2.3 Mechanismen und Funktionen durch Testfälle testen

Die Testfälle, die in Abschnitt 4.3.3 vorgestellt werden, sind jeweils in separaten Dateien implementiert. Diese sind nach dem Schema *testCase*∗*.c* benannt. Wobei das ∗ durch den Namen des jeweiligen Testfalles ersetzt wird. Die Dateien befinden sich im Ordner *testCase*.
In jedem Testfall befinden sich mindestens die Bestandteile wie in Listing 5.2 für den Testfall "Minimal" dargestellt. In *testCaseMinimalDependencies* befinden sich die Namen der Funktionen, die für das erfolgreiche Ausführen dieses Testfalles nötig sind. Zu Beginn jedes Testfalles wird geprüft, ob alle diese Funktionen unterstützt werden. Dazu wird die Liste, die in dem in Abschnitt 5.2.1 vorgestellten Vorgang entstanden ist, nach dem entsprechenden Eintrag durchsucht. Ist das Feld *impl* einer der zu prüfenden Funktionen in der Liste auf null gesetzt, wird der Testfall nicht ausgeführt.

```
const char *testCaseMinimalDependencies[] = {"C_Initialize", "
    C_GetInfo", "C_Finalize", NULL};
int testCaseMinimal();
```

Listing 5.2: *Struktur der Implementationsdatei eines Testfalles am Beispiel: Minimal*

Funktionen, die von allen Testfällen genutzt werden können, befinden sich in den Dateien *testCaseHelper.c* und *pkcs11OutputConstants.c*. In der *pkcs11OutputConstants.c* Datei befinden sich vor allem Funktionen, um aus den Zahlenwerten von Cryptoki die entsprechenden Konstantennamen zu ermitteln.
Zwischen den Testfällen bestehen keine Abhängigkeiten. Dies hat zur Folge, dass bei jedem Testfall Cryptoki wieder initialisiert und finalisiert wird. Dazu ergibt sich der Vorteil, dass die einzelnen Testfälle entfernt bzw. weiterentwickelt werden können, ohne dass die anderen beeinflusst werden.
Bei jedem Testfall werden alle aktiven Token durchlaufen. Das bedeutet, dass z.B. zwei Token gleichzeitig getestet werden könnten. Dies ist jedoch noch nicht getestet worden, da zur Entwicklungszeit keine entsprechenden Token und Lesegeräte zur Verfügung standen.
Im Folgenden werden die Implementierungen der Testfälle kurz vorgestellt und auf Besonderheiten hingewiesen.

Testfall 1: Minimal

Dateiname: testCaseMinimal.c
In diesem Testfall wird geprüft, ob die Initalisiserung von Cryptoki und die Ausführung von *C_GetInfo* funktioniert. Dazu werden die Funktionen in folgender Reihenfolge ausgeführt.

- *C_Finalize*
- *C_Initialize*
- *C_GetInfo*
- *C_Finalize*
- *C_Finalize*

So wird geprüft, ob Cryptoki schon initialisiert ist, sich initialisieren lässt, sich eine Funktion nach der Initialisierung ausführen lässt und ob das Beenden von Cryptoki funktioniert.

Testfall 2: SlotInformation

Dateiname: testCaseSlotInformation.c
In diesem Testfall werden die slotbezogenen Funktionen getestet. Dabei werden, wie im Entwurf geplant die Funktion *C_GetSlotList* und auf den so ermittelten Slots die Funktion *C_GetSlotInfo* aufgerufen. Da TCH die kryptographische Hardware und nicht das Modul testet, werden ab diesem Testfall nur noch Slots mit einem Token getestet.

Testfall 3: TokenInformation

Dateiname: testCaseTokenInformation.c
In diesem Testfall wird über alle mittels *C_GetSlotList* ermittelten Token iteriert und *C_GetMechanismList* und *C_GetTokenInfo* ausgeführt. Sollte beim Starten von TCH nicht alle Informationen über das Token ausgegeben werden, wie in Abschnitt 4.3.2 entworfen, so lässt sich hier die Ursache genauer ermitteln.

Testfall 4: CompareMechanismLists

Dateiname: testCaseCompareMechanismLists.c
Ziel des entworfenen Testfalls ist es, zu überprüfen, ob die Mechanismen die mittels *C_GetMechanismList* ermittelt wurden auch über *C_GetMechanismInfo* bestätigt werden können. Um darüber hinaus festzustellen, ob es Mechanismen gibt, die nicht über *C_GetMechanismList* übergeben werden, aber trotzdem unterstützt werden, teilt sich dieser Testfall in zwei Phasen auf.
Zuerst wird über alle definierten Mechanismen iteriert. Um über alle Mechanismen iterieren zu können, gibt es die Liste $MECHANISM_LIST$, die neben dem Zahlenwert des Mechanismus auch den Konstantennamen enthält. Es wird für jedes Element dieser Liste *C_GetMechanismInfos* aufgerufen. Wenn der Rückgabewert nicht *CKR_MECHANISM_INVALID* ist, dann wird geschaut, ob der Mechanismus in der Liste, die über *C_GetMechanismList* ermittelt wurde, enthalten ist.
Der zweite Teil dieses Testfalles ruft für alle Mechanismen, die in der Liste, die über *C_GetMechanismList* ermittelt wurde, *C_GetMechanismInfo* auf. Dabei erkannte Inkonsistenzen werden angezeigt. Da die Testfälle untereinander keinen Einfluss nehmen sollen, wird die in Abschnitt 5.2.2 beschriebene Liste der Mechanismen nicht manipuliert. Dies geschieht auch aus dem Grund, da nicht klar ist, welche der beiden Funktionen nun die korrekten Ergebnisse liefert.

Testfall 5: SlotEvents

Dateiname: testCaseSlotEvents.c
Dieser Testfall verwendet wie entworfen die Funktionen *C_GetSlotInfo* und *C_WaitForSlotEvent*, um festzustellen, ob ein Token entfernt und eingesteckt werden kann und dies richtig erkannt wird. Voraussetzung ist, dass das Flagfeld, das durch *C_GetSlotInfo* ermittelt wird, erkennen lässt, dass es sich um ein entfernbares Token handelt. Ist das Flag nicht gesetzt, wird der Testfall abgebrochen. Dieser Testfall prüft genau genommen das Zusammenspiel von Token und Modul und nicht das Token. Hier muss insbesondere unterschieden werden zwischen der physikalischen Transportabilität des Tokens und dem Funktionsumfang des Modules. Das USB-Token, wie in Abschnitt 6.1 vorgestellt, ist zwar tragbar, aber das Modul gibt an, dass das Token nicht entfernbar ist.

Testfall 6: SingleSessionWithUserLogin

Dateiname: testCaseSingleSessionWithUserLogin.c
In diesem Testfall wird eine Session mit *C_OpenSession* gestartet, mit *C_GetSessionInfo* der aktuelle Status ermittelt und mit *C_CloseSession* die Session wieder geschlossen. Dabei wird abhängig von den Parametern, -u bzw. -s, wie in Abschnitt 5.1.3 erläutert, der User bzw. der SO eingeloggt und ausgeloggt und der Status der Session beobachtet.

Testfall 7: MultipleSessionWithUserLogin

Dateiname: testCaseMultipleSessionWithUserLogin.c
In diesem Testfall werden im Gegensatz zum vorherigen, zwei Sessions erstellt und am Ende mit *C_CloseAllSession* geschlossen. Dabei wird auch geprüft, ob der Loginstatus sich in den Flagfeldern beider Sessions ändert, wenn sich mit einer Session eingeloggt, bzw. ausgeloggt wird. Für das vollständige Ausführen dieses Testfalles ist es erforderlich, dass sowohl die User als auch die SO PIN übergeben wurden.

Testfall 8: PreserveOperationState

Dateiname: testCasePreserveOperationState.c
Dieser Testfall wurde in der aktuellen Version von TCH nur in einer verringerten Version gegebenüber dem Entwurf implementiert. Grund ist, dass leider keine Geräte zur Verfügung stehen, die ein experimentieren ermöglichen. Der Entwurf wurde im Abschnitt 4.3.3 vorgestellt.

Testfall 9: AdminToken

Dateiname: testCaseAdminToken.c
Wurde der Parameter -i, -s und -u beim Start übergeben, so startet dieser Testfall. Es

wird erst *C_InitToken* mit der -s übergebenen PIN aufgerufen, um das Token neu bzw. erstmalig zu initialisieren. Anschließend wird mit *C_InitPIN* nach erfolgreichem Login des SOs die User PIN, also das, was mit -u übergeben wurde, gesetzt.

Testfall 10: ChangePin

Dateiname: testCaseChangePin.c
Für diesen Testfall muss der Parameter -o gesetzt sein und mindestens eine PIN mit -u oder -s übergeben worden sein. Abhängig, ob eine User bzw. SO PIN übergeben wurde, wird ein entsprechender Login ausgeführt und danach *C_ChangePin* mit einem neuem Passwort aufgerufen. Ist das Setzen erfolgreich, wird ein erneuter Login mit diesem ausgeführt und das Passwort wieder auf das alte zurückgesetzt. Wenn das Setzen des Passwortes nicht erfolgreich, aber der Login vorher erfolgreich war, wird dies angezeigt. Scheitert der Login wird TCH, wie in Abschnitt 5.1.3 erwähnt, beendet.

Testfall 11: Objecthandling

Dateiname: testCaseObjectHandling.c
In diesem Testfall wird eine Session mit Schreibberechtigung geöffnet und versucht, ein Objekt von der Klasse data zu erzeugen, dessen Label neu zu setzen, diese Information zu bestätigen, die Objektgröße zu ermitteln, zu kopieren und anschließend zu löschen. Die Funktionen *C_GetAttributeValue*, *C_SetAttributeValue*, *C_GetObjectSize* und *C_CopyObject* sind optional. Ob diese unterstützt werden, wird vor dem Aufruf festgestellt und die entsprechenden Funktionen ausgeführt oder übersprungen.

Testfall 12: ObjectSearch

Dateiname: testCaeObjectSearch.c
Wenn zu Beginn der Ausführung von TCH die Anzahl der auf dem Token gespeicherten Objekte angezeigt wird, dann funktioniert die Objektsuche. In diesem Testfall werden Objekte erzeugt und anschließend nach Ihnen gesucht. So kann überprüft werden, ob die Anzahl der gefundenen Objekte mit der Erzeugten übereinstimmt.

Testfall 13: DeAndEncryptionSimple

Dateiname: testCaseDeAndEncryptionSimple.c
In diesem Testfall soll die Ver- und Entschlüsselung von einzelnen Datensätzen getestet werden, also mit *C_EncryptInit* und folgendem *C_Encrypt* bzw. für das Entschlüsseln *C_DecryptInit* und *C_Decrypt*. Dazu wird über alle unterstützten Mechanismen iteriert. Unterstützt der Mechanismus sowohl Ver- als auch Entschlüsselung und unterstützt das Gerät die Erzeugung eines entsprechenden Schlüssels, so wird der Schlüssel erzeugt und verschieden lange zufällig erzeugte Zeichenketten erst ver- und anschließend entschlüsselt. Dabei wird die entsprechende Zeit mit der Funktion *gettimeofday* ermittelt.

Testfall 14: DeAndEncryptionMultiple

Dateiname: testCaseDeAndEncryptionMultiple.c
Dieser Testfall läuft fast gleich ab wie der vorherige. Anstatt die zufällig erzeugten Zeichenketten in einem Funktionsaufruf zu bearbeiten, wird diese geteilt und der Verschlüsselungsvorgang mit *C_EncryptUpdate* und *C_EncryptFinal* bzw. der Entschlüsselungsvorgang *C_DecryptUpdate* und *C_DecryptFinal* ausgeführt.

Testfall 15: DigestSimple

Dateiname: testCaseDigistSimple.c
In diesem Testfall wird über alle unterstützten Mechanismen iteriert. Unterstützt der Mechanismus die Digest Funktion, so wird erst *C_DigestInit* und anschließend *C_Digest* ausgeführt. Als zu verarbeitende Daten werden zufällig erzeugte Zeichenketten verwendet. Auch hier wird die Zeit pro Vorgang aufgezeichnet. *C_DigestKey* wird bei jeder Iteration, auf einem mit *CKM_DES_KEY_GEN* erzeugten Schlüssel, ausgeführt. Ein Testen des Digestvorgangs für alle verfügbaren Schüsselgenerierungsmechanismen ist, um zu testen ob die Funktion korrekt ausgeführt werden kann, nicht nötig.

Testfall 16: DigestMultiple

Dateiname: testCaseDigistMultiple.c
In diesem Testfall wird im Gegensatz zum vorherigen die generierte Zeichenkette geteilt und anschließend mit wiederholten Aufrufen von *C_DigestUpdate* und einem abschließendem Aufruf von *C_DigestFinal* abgearbeitet.

Testfall 17: SignAndVerifySimple

Dateiname: testCaseSignAndVerifySimple.c
In diesem Testfall wird über alle unterstützten Mechanismen iteriert und alle Mechanismen, die sowohl das Signieren als auch das Verifizieren unterstützen, werden mit *C_SignInit* und *C_Sign* und anschließendem *C_VerifyInit* und *C_Verify* ausgeführt. Signiert werden dabei zufällige Zeichenketten verschiedener Länge, die im Anschluss verifiziert werden.

Testfall 18: SignAndVerifyMultiple

Dateiname: testCaseSignAndVerifyMultiple.c
In diesem Testfall wird anstatt die Zeichenkette in einem Funktionsaufruf mit *C_Sign* bzw. *C_Verify* zu verarbeiten, diese geteilt, so dass *C_VerifyUpdate* und *C_VerifyFinal* bzw. *C_SignUpdate* und *C_SignFinal* getestet werden können.

Testfall 19: SignAndVerifyRecover

Dateiname: testCaseSignAndVerifyRecover.c
Die Datei für diesen Testfall ist angelegt und eingebunden. Leider gab es während der Arbeit mangels fehlender Unterstützung der zur Verfügung stehenden Token keine Möglichkeit, diesen Testfall an praktischen Beispielen zu entwickeln.

Testfall 20: DualPurposeFunctions

Dateiname: testCaseDualPurposeFunctions.c
Die Datei für diesen Testfall ist angelegt und eingebunden. Leider gab es während der Arbeit mangels fehlender Unterstützung der zur Verfügung stehenden Token keine Möglichkeit, diesen Testfall an praktischen Beispielen zu entwickeln.

Testfall 21: SingleKeyGeneration

Dateiname: testCaseSingleKeyGeneration.c
In diesem Testfall wird über alle unterstützten Mechanismen iteriert. Bei allen Mechanismen, die zur Schlüsselerzeugung genutzt werden können, wird festgestellt, ob TCH für den Mechanismus die entsprechenden Parameter bereitstellen kann. Ist dies der Fall, wird *C_GenerateKey* ausgeführt.
Eine Anmerkung zur Schlüsselerzeugung befindet sich in Abschnitt 5.3.1.

Testfall 22: KeyPairGeneration

Dateiname: testCaseKeyPairGeneration.c
In diesem Testfall wird die Erzeugung von Schlüsselpaaren mittels der *C_GenerateKeyPair* getestet.
Eine Anmerkung zur Schlüsselerzeugung befindet sich in Abschnitt 5.3.1.

Testfall 23: WrapAndUnwrapKey

Dateiname: testCaseWrapAndUnwrapKey.c
Die Datei für diesen Testfall ist angelegt und eingebunden. Leider gab es während der Arbeit mangels fehlender Unterstützung der zur Verfügung stehenden Token keine Möglichkeit, diesen Testfall an praktischen Beispielen zu entwickeln.

Testfall 24: DeriveKey

Dateiname: testCaseWrapAndUnwrapKey.c
Dieser Testfall sollte das Ableiten eines Schlüssels testen. Da leider während der Entstehung dieser Arbeit kein Token zur Verfügung stand, welches einen entsprechenden Mechanismus unterstützt, konnte nicht an einem praktischen Beispiel entwickelt werden. Der skizzierte Ablauf befindet sich in der vorbereiteten Datei.

Testfall 25: GenerateRandom

Dateiname: testCaseGenerateRandom.c
In diesem Testfall wird, wenn unterstützt, *C_SeedRandom* aufgerufen und anschließend *C_GenerateRandom* mehrfach mit verschiedenen Längen aufgerufen. Die Zeit der Zufallswertgenerierung wird dabei mitgezeichnet. Um anschließend z.B. eine weitere statistische Auswertung der Werte durch den Anwender zu ermöglichen, werden diese in eine Datei im TCH Ordner geschrieben.

5.3 Anmerkungen

5.3.1 Schlüsselerzeugung

Für Mechanismen sind für die Schlüsselerzeugung verschiedene Parameter in PKCS #11 definiert. Bei dem Erzeugen eines RSA Schlüsselpaares wären das z.B. Parameter wie die Schlüssellänge oder der Exponent. In TCH sind nicht alle definierten Mechanismen mit den entsprechenden Parametern hinterlegt. Es wurden diejenigen umgesetzt, welche von den zur Verfügung stehenden Token unterstützt werden. Soll ein Schlüssel erzeugt werden, so wird erst festgestellt, ob es sich um einen Mechanismus für *C_GenerateKey* oder für *C_GenerateKeyPair* handelt. Sind die entsprechenden Parameter vorhanden, so wird der Schlüssel erzeugt.

6 Evaluation

In diesem Kapitel soll mithilfe der erarbeiteten Software kryptographische Hardware getestet und verglichen und somit TCH evaluiert werden.

6.1 Testaufbau

Für die Tests wurde ein IDProtect Key der Firma athena [16], mit einem USB Interface, und die Smartcard IDPrime .NET 510 der Firma Gemalto [10] verwendet. Diese werden in den folgenden Abschnitten verkürzt nur Key bzw. Stick für den IDProtect Key und Smartcard für die IDPrime Smartcard genannt. Abbildung 4 zeigt ein Bild von diesen. Beide Token befinden sich preislich in einem Rahmen von etwa 20€ bis 30€ und unterstützen PKCS #11. Es wurden die entsprechenden über den Hersteller bzw. den Lieferanten zu erlangende PKCS #11 Module für Linux verwendet.

(a) IDProtect Key (b) Smartcard IDPrime

Abbildung 4: *Fotos der beiden Testtokens*

Kompiliert wurde TCH mit $gcc\ -Wall\ -o\ \text{“}tch\text{“}\ \text{“}tch.c\text{“}\ -ldl\ -g$. Das Argument *ldl* ist notwendig, da TCH mit dynamisch gelinkten Objekten -den Modulen- arbeitet. Das verwendete Betriebssystem ist das schon in Abschnitt 5.1.1 erwähnte Ubuntu 14.04 LTS mit $gcc(Ubuntu4.8.2-19ubuntu1)4.8.2$.

Die Token wurden mit TCH, mit den Parametern -o -v -u, getestet. Dabei kam es bei dem Stick zu einem Stopp des Programmes, wie in Abschnitt 6.2.3 Testfall 5 erwähnt

wird, der TCH an einer vollständigen Durchführung hinderte. Um die Tests fortsetzen zu können, wurden die entsprechenden Zeilen im Quellcode auskommentiert und TCH neu kompiliert.

6.2 Testergebnisse

In diesem Abschnitt werden einige Ausgaben von TCH unter der Verwendung der beiden in Abschnitt 6.1 genannten Token mit den entsprechenden Modulen aufgeführt. Gegliedert ist dies nach dem in Abschnitt 4.3 vorgestellten Ablauf. Die vollständigen Ausgaben von TCH zu den Geräten befinden sich im Anhang 3 und 4.

6.2.1 Funktionen auf Implementation testen

Tabelle 17 zeigt die jeweils nicht-implementierten Funktionen von Cryptoki die sich nach dem in Abschnitt 5.2.1 vorgestellten Verfahren ergeben. Wie zu erkennen ist, unterstützten beide Token die Mehrzweckfunktionen -wie in Abschnitt 2.3.2 vorgestellt- nicht. Auch die Funktionen *C_GetOperationState*, *C_SetOperationState*, *C_CopyObject* und *C_WrapKey* werden von beiden Geräten nicht unterstützt. Darüber hinaus unterstützt die Smartcard weitere 13 Funktionen weniger als der USB-Stick.
Während des Implementationstests kam es bei dem IDProtect Key zu Abstürzen von TCH. Vermutlich überprüft das Modul die übergebenen Argumente nicht ausreichend und versucht so auf eine Stelle im Speicher zuzugreifen, auf die sie keinen Zugriff hat. Um ein Weiterführen der Tests zu ermöglichen, wurde für die Tests mit dem IDProtect Key die entsprechenden Implementationstests der Funktionen *C_Sign*, *C_SignFinal* und *C_SignRecover* auskommentiert. In zukünftigen Versionen von TCH erscheint es aus diesem Grund sinnvoll, die Implementationstests flexibler zu gestalten, sodass einzelne Funktionen geprüft bzw. übersprungen werden können.

IDProtect Key	IDPrime Smartcard
C_GetOperationState, *C_SetOperationState*, *C_CopyObject*, *C_DigestEncryptUpdate*, *C_DecryptDigestUpdate*, *C_SignEncryptUpdate*, *C_DecryptVerifyUpdate*, *C_WrapKey*	*C_GetOperationState*, *C_SetOperationState*, *C_CopyObject*, *C_GetObjectSize*, *C_EncryptUpdate*, *C_EncryptFinal*, *C_DecryptUpdate*, *C_DecryptFinal*, *C_DigestKey*, *C_SignRecoverInit*, *C_SignRecover*, *C_VerifyRecoverInit*, *C_VerifyRecover*, *C_DigestEncryptUpdate*, *C_DecryptDigestUpdate*, *C_SignEncryptUpdate*, *C_DecryptVerifyUpdate*, *C_GenerateKey*, *C_WrapKey*, *C_UnwrapKey*, *C_DeriveKey*

Tabelle 17: *Übersicht der nicht-implementierten Funktionen von Cryptoki auf den Token*

6.2.2 Informationen auslesen

Im folgenden Abschnitt werden einige Informationen der Token dargelegt, die sich durch das Verwenden der entsprechenden Funktionen wie in Abschnitt 4.3.2 dargelegt ergeben.

Metainformationen auslesen

Die Funktionen *C_GetSlotInfo*, *C_GetTokenInfo*, *C_GetMechanismList* und *C_GeMechanismInfo* sind nach der Tabelle 17 in Abschnitt 6.2.1 implementiert und liefern wie in Anhang 3 bzw. 4 ersichtlich Informationen über das Token.
An diesen Informationen kann beispielsweise direkt abgelesen werden, ob es sich um ein Hardwaretoken handelt oder ob das Token selber Zufallswerte erzeugen kann. Dies ist an den Flags erkennbar, die über *C_GetSlotInfo* und *C_GetTokenInfo* ermittelt werden. Dazu lassen sich auch Beschränkungen der Hardware wie die Speicherkapazität oder die Möglichkeiten der PIN-Länge ablesen.
Die beiden Token unterstützen unterschiedlich viele Mechanismen. Wie bereits in Tabelle 17 in Abschnitt 6.2.1 erkennbar ist, unterstützt die Smartcard die Funktion *C_GenerateKey* nicht. Es findet sich auch kein entsprechender Mechanismus in der Liste der unterstützten Mechanismen.
Den Mechanismus zum Erzeugen von RSA Schlüsseln (*CKM_RSA_PKCS_KEY_PAIR_GEN*) und einige Hash-Algorithmen wie den MD5 und SHA1 unterstützten beide Geräte. Auch hier finden sich, betrachtet man die Ausgaben im Detail, einige Unterschiede. So unterstützt z.B. der IDProtect Key nicht nur das Hashen mit dem SHA1, sondern auch das Signieren und Verifizieren von Daten. Anhand dieser Ausgabe ist es möglich, Geräte für den Anwendungsfall auszuwählen. Wenn ein Token an Cryptoki angebunden werden soll, so würde ein Blick in das Datenblatt von dem IDProtect Key potentielle Irrschlüsse ermöglichen. So ist dort unter den unterstützten Hash-Algorithmen der GOST 3411 aufgeführt. In Cryptoki 2.2 ist dieser nicht definiert.
Auch wird über den Blick auf die Ausgaben von TCH klar, welche Mechanismen tatsächlich auf der Hardware ausgeführt werden. Das Flag mit dem Namen *CKF_HW* ist überall dort aufgeführt, wo der Algorithmus laut Token auf dem Token selber ausgeführt wird.
Inwieweit die Metainformationen einen brauchbaren Informations- und Wahrheitsgehalt haben ist grundsätzlich in Frage zu stellen.
So hat der IDProtect Key nach Ausgabe der Informationen 18446744073709551615 -in hexadezimal FFFFFFFFFFFFFFFF- öffentlichen und privaten Speicher. Durch den PKCS #11 Standard geht hervor, dass es nicht definiert ist, wie diese Zahl zu interpretieren ist [28]. Das Datenblatt spricht von 72k EEPROM. Welche der beiden Werte dem realen freien Speicher entspricht ist nicht erkennbar. Wie in Experimenten festgestellt wurde, ändert sich der Wert des freien Speichers bei der Smartcard nicht, wenn Objekte hinzukommen.

Objekte auslesen

Im Anschluss an die Ermittlung der Metainformationen werden die auf dem Token befindlichen Objekte angezeigt. Diese werden wie in Abschnitt 4.3.2 beschrieben nach den Objektarten separiert gesucht. Beim IDProtect Key und der Smartcard zeigte sich nach einigen Versuchen, dass die Objekte unabhängig von der Objektart gefunden werden. Wenn nach einem key-Objekt gesucht wurde, wurde auch das data-Objekt angezeigt. Dies wird allerdings nicht durch ein einmaliges Ausführen von TCH sichtbar. Ein möglicher Testfall, um dies sichtbar zu machen, wäre das Erstellen von einzelnen Objekten und das anschließende Suchen nach Objekten aller Arten. Der Testfall 12: Objektsearch wie in Abschnitt 5.2.3 erläutert, könnte um diesen Vorgang erweitert werden.

6.2.3 Ergebnisse der Testfälle

Im folgenden Abschnitt werden die Ergebnisse der einzelnen Testfälle, wie sie in Abschnitt 5.2.3 erklärt wurden, aufgeführt.

Testfall 1 - 3: Minimal, TokenInformationSlotInformation und TokenInformation

Diese Testfälle wurden von beiden Token erfolgreich ausgeführt. Es konnten keine unerwarteten Rückgabewerte beobachtet werden.
Auffällig ist lediglich, dass das Modul der Smartcard fünf Slots kennt. Mit einem Token inne wird allerdings, wie erwartet, nur ein Slot, mit der Nummer 0, aufgeführt.

Testfall 4: CompareMechanismLists

Das Vergleichen der ermittelten Mechanismenlisten wie in Abschnitt 5.2.3 beschrieben führt zu dem Ergebnis, dass beide Listen kongruent sind. Das bedeutet, dass der Aufruf von *C_GetMechanismList* verwendet werden kann, um eine vollständige Liste der unterstützten Mechanismen, zu denen auch Informationen bereitgestellt werden können, zu erhalten.

Testfall 5: SlotEvents

Wie an den Flags zu erkennen ist, handelt es sich bei der Smartcard und dem IDProtect Key um entfernbare Geräte (*CKF_REMOVABLE_DEVICE*).
Das Modul der Smartcard erkennt das Ein- und Ausstecken jener. Beim Key wird dies nicht erkannt. Dies führt dazu, dass der Vorgang nicht weitergeführt werden kann, da auf ein Event gewartet wird, welches nicht ausgelöst wird. Der Testfall musste an dieser Stelle manuell abgebrochen werden. Die Funktion *C_WaitForSlotEvents* kann mit dem IDProtect Key also nicht verwendet werden, um ein Entfernen des Gerätes zu bemerken.
In der aktuellen Version von TCH ist es nicht möglich, einzelne Testfälle zu starten.

In Szenarien wie dem hier vorliegenden, wird aber deutlich, dass dies ein sinnvolles Feature ist. In dem vorliegenden Evaluationslauf wurde TCH ohne den Aufruf des Testfalles 5 kompiliert, um ein Durchlaufen zu ermöglichen.

Testfall 6 & 7: SingleSessionWithUserLogin und MultipleSessionWithUserLogin

Die beiden Testfälle 6 und 7 konnten vollständig ausgeführt werden. Bei einer Session die ohne *CKF_SERIAL_SESSION* Flag aufgerufen wurde, wurde wie erwartet der Rückgabewert *CKR_SESSION_PARALLEL_NOT_SUPPORTED* zurückgegeben. Zudem ist ersichtlich, dass der Sessionstatus sich bei zwei Sessions nach einem Login bzw. Logout bei beiden Sessions ändert.

Testfall 8: PreserveOperationState

Dieser Testfall wurde aufgrund von Funktionsabhängigkeiten bei beiden Token übersprungen. Die Funktionen *C_GetOperationState* und *C_SetOperationState* werden von beiden Token als nicht unterstützt angegeben.

Testfall 9: AdminToken

Der Testfall 9 kann mit den vorliegenden Token nicht mehr vollständig durchlaufen werden. Der IDProtect Key hat den SO gesperrt und für die Smartcard scheint die vom Händler genannte SO PIN nicht zu stimmen.

Testfall 10: ChangePin

Dass der Loginvorgang funktioniert, wurde bereits in Testfall 6 bzw. 7 sichtbar. Der IDProtect Key gibt bei dem Aufruf von *C_SetPIN* den Rückgabewert *CKR_PIN_INVALID* zurück. Anhand des Rückgabewertes des Keys, *CKR_PIN_INVALID*, lässt sich leider nicht genau erschließen, wo ein Problem entstehen könnte. Die PIN, die versucht wird zu setzen(0000), besteht aus den gleichen Zeichen wie der aktuelle PIN(00000).
Die Smartcard hingegen ändert die PIN des Users und ermöglicht einen anschließenden Login und das Zurücksetzen der PIN.

Testfall 11: Objecthandling

Wie in Tabelle 17 in Abschnitt 6.2.1 ersichtlich ist, unterstützen die beiden Token unterschiedliche Objektverwaltungsfunktionen. So unterstützten beide die Funktion *C_CopyObject* nicht. Der Key unterstützt im Gegensatz zur Smartcard noch die Funktion *C_GetObjectSize*. Alle unterstützten Funktionen konnten erfolgreich ausgeführt werden und der Inhalt des erhaltenen Attributes über die Funktion *C_GetAttributeValue* ist nach dem Setzen jenes wie erwartet. Auf beiden Token können also Datenobjekte erstellt, bearbeitet und zerstört werden.

Testfall 12: ObjectSearch

Bei beiden Token konnten drei Objekte erstellt und anschließend gefunden werden. Dabei traten keine unerwarteten Ereignisse auf.

Testfall 13: DeAndEncryptionSimple

Die Ergebnisse der beiden Geräte bei diesem Testfall weichen stark voneinander ab. Dies ergibt sich zum Teil aus den unterschiedlich unterstützten Mechanismen, aber auch aus den Umgang mit Parametern.
Auf dem Key kann mit einigen Mechanismen erfolgreich ver- und entschlüsselt werden. Die Tabelle 18 zeigt die entsprechende Zeiten bei zumindest teilweise erfolgreichen Verschlüsselungsvorgängen. Die Zeiten sind die Mittelwerte der Werte die sich durch fünfmaliges Ausführen von TCH ergeben haben. Es scheitern lediglich verschiedene Längen -vermutlich an der Beschränkung der Algorithmen.
Die Smartcard kann die entsprechenden Schlüssel generieren, gibt aber beim Aufruf von *C_Encrypt* den Wert *CKR_OBJECT_HANDLE_INVALID* zurück. Dieser Rückgabewert ist in PKCS #11 nicht für diese Funktion vorgesehen. Da in dem Aufruf von *C_Encrypt* kein Objekt selber übergeben wird, lässt sich vermuten, dass beim Aufruf von *C_EncryptInit* der eigentliche Fehler schon entstanden ist.

Mechanismus/Datenlänge in CK_BYTE	16	1048576	33554432
CKM_RSA_PKCS	0,018108 Sek.	-	-
CKM_RSA_X_509	0,0145714 Sek.	-	-
CKM_DES_ECB	0,0008762 Sek.	0,2233154 Sek.	6,971447 Sek.
CKM_DES3_ECB	0,001091 Sek.	0,6480784 Sek.	20,7494424 Sek.
CKM_DES_CBC	0,000503 Sek.	0,2241038 Sek.	7,1780158 Sek.
CKM_DES3_CBC	0,0010884 Sek.	0,6533616 Sek.	20,8042742 Sek.

Tabelle 18: *Zeitwerde der Verschlüsselung von Daten mittels verschiedener Mechanismen mit dem IDProtect Key*

Testfall 14: DeAndEncryptionMultiple

Dieser Testfall wird von der Smartcard übersprungen, da die Funktion *C_EncryptUpdate* fehlt. Der Key startet zwar den Testvorgang, kann aber keine Daten erfolgreich verschlüsseln. Mangels weiterer Testgeräte kann leider nicht ausgeschlossen werden, dass es sich hier um einen Fehler in TCH handelt.
Die Rückgabewerte sind allerdings unabhängig davon unerwartet. Der Rückgabewert *CKR_BUFFER_TOO_SMALL* wird von *C_EncryptUpdate* geliefert. Da die in Abschnitt 2.3.2 genannte Konvention verwendet wird, dürfte dieser Fall nicht eintreten, es sei denn, die Schätzung ist falsch oder der Rückgabewert ist fehlerhaft.

Testfall 15 & 16: DigestSimple und DigestMultiple

Beide Geräte passieren diese Testfälle ohne unerwartete Rückgabewerte. Das Hashen eines 64-Zeichen-langen Strings ist mit allen unterstützten Mechanismen erfolgreich. An den Flags der Mechanismen lässt sich erkennen, dass beide Geräte kein Mechanismus mit gleichzeitig gesetzten *CKF_DIGEST* und *CKF_HW* Flag unterstützen. Dies zeigt an, dass die Digestvorgänge nicht vollständig auf dem Token ablaufen. Eine Zeitmessung würde also nicht die Performance des Tokens messen, sondern die des Rechners, auf dem TCH läuft.
Die Smartcard überspringt den Test von *C_DigestKey*, da die Funktion nicht unterstützt wird.

Testfall 17 & 18: SignAndVerifySimple und SignAndVerifyMultiple

Beide Token unterstützen die für die beiden Testfälle notwendigen Funktionen. Sowohl der Key als auch die Smartcard signieren zumindest einmal erfolgreich Daten mit den Mechanismen: *CKM_RSA_PKCS*, *CKM_RSA_X_*509, *CKM_SHA*1*_RSA_PKCS* und *CKM_SHA*256*_RSA_PKCS*. Darüber war mit der Smartcard ein Signiervorgang erfolgreich mit dem Mechanismus *CKM_MD*5*_RSA_PKCS*. Der Key führte weitere Mechanismen wie *CKM_SHA*384*_RSA_PKCS* und *CKM_SHA*512*_RSA_PKCS* erfolgreich durch.
Die Karte gibt bei Testfall 17 bei verschiedenen Mechanismen den Rückgabewert *CKR_OBJECT_HANDLE_INVALID* zurück. Da auch diese Funktion wie auch die Funktion *C_Encrypt* in Testfall 13 kein Objekt entgegen nimmt, ist die Ursache der teilweise nicht funktionierenden Vorgänge unklar. Vermutlich liegt es an dem Schlüssel, der zum Signieren und Verifizieren verwendet wird. Dies sollte allerdings bei der Initialisierung z.B. bei *C_SignInit* durch einen entsprechenden Rückgabewert deutlich gemacht werden.
Da die Initialisierung des Signier- bzw. Verifiziervorgangs erfolgreich durchgeführt wird, muss nach einem Signier- bzw. Verifiziervorgang die Session beendet werden, bevor eine weitere kryptographische Funktion gestartet werden kann. Der Rückgabewert *CKR_OPERATION_ACTIVE* könnte ansonsten bei dem Initialisieren eines neuen Vorganges erhalten werden.
Nach Tests die in der Entwicklungsphase von TCH durchgeführt wurden, wurde bemerkt, dass die Smartcard genau dieses Verhalten aufweist. Der Key startet im Gegensatz dazu neue Vorgänge wie z.B. das Signieren und Verifizieren, auch wenn der vorherige nicht vollständig beendet wurde.
Das Signieren und Verifizieren in mehreren Durchläufen wird von einigen Mechanismen bzw. den entsprechenden Schlüsseln nicht unterstützt. Dies wird vom Key auch wie erwartet mit einem *CKR_KEY_FUNCTION_NOT_PERMITTED* zurückgegeben.

Testfall 19: SignAndVerifyRecover

Dieser Testfall wurde aufgrund von Funktionsabängigkeiten von der Smartcard überspringen. Die Funktionen *C_SignRecoverInit*, *C_SignRecover*, *C_VerifyRecoverInit*

und *C_VerifyRecover* werden als nicht unterstützt angegeben. Der IDProtect Key unterstützt zwar die Funktionen, dafür keinen entsprechenden Mechanismus, dies ist an dem fehlenden *CKF_VERIFY_RECOVER* bzw. *CKF_SIGN_RECOVER* Flag bei den unterstützten Mechanismen zu erkennen.

Testfall 20: DualPurposeFunctions

Dieser Testfall wurde aufgrund von Funktionsabängigkeiten bei beiden Token übersprungen. Die Funktionen *C_DigestEncryptUpdate*, *C_DecryptDigestUpdate*, *C_SignEncryptUpdate* und *C_DecryptVerifyUpdate* werden von beiden Token als nicht unterstützt angegeben.

Testfall 21: SingleKeyGeneration

Dieser Testfall wird von der Smartcard übersprungen. Sie unterstützt keinen entsprechenden Mechanismus und auch nicht die Funktion *C_GenerateKey*.
Der Key liefert auf die Mechanismen *CKM_DES_KEY_GEN*, *CKM_DES2_KEY_GEN* und *CKM_DES3_KEY_GEN* ein *CKR_OK* zurück. Der Mechanismus *CKM_AES_KEY_GEN* liefert ein *CKR_ATTRIBUTE_VALUE_INVALID*. Dies lässt darauf schließen, dass das Template, welches -zur Schlüsselgenerierung verwendet- ein Attribut mit einem fehlerhaften Wert enthält. Aber auch mit einem Wechseln der Werte, konnte diese Fehlermeldung nicht verändert werden.

Testfall 22: KeyPairGeneration

Beide Token unterstützen den Mechanismus *CKM_RSA_PKCS_KEY_PAIR_GEN* der für diesen Testfall in Frage kommt. Und beide Token führen erfolgreich die Funktion *C_GenerateKeyPair* mit dem genannten Mechanismus aus.
An dieser Stelle ließen sich noch viele weitere Versuche anbringen. Besonders mit den Schlüssellängen und inwieweit man den generierten Schlüssel manipulieren, exportieren und verwenden kann.

Testfall 23: WrapAndUnwrapKey

Dieser Testfall wurde aufgrund von Funktionsabängigkeiten bei beiden Token übersprungen. Laut Mechanismenliste unterstützt der IDProtect Key aber zumindest einen Mechanismus zum Entpacken eines Schlüssels.

Testfall 24: DeriveKey

Dieser Testfall wurde aufgrund von Funktionsabängigkeiten bei beiden Token übersprungen. Wie sich an der Tabelle 17 in Abschnitt 6.2.1 ablesen lässt, unterstützt der

IDProtect Key zwar die Funktion *C_DeriveKey*, jedoch keinen Mechanismus mit dem entsprechenden Flag *CKA_DERIVE*.

Testfall 25: GenerateRandom

Beide Token unterstützen die Funktionen *C_GenerateRandom* und *C_SeedRandom*. *C_SeedRandom* gibt bei beiden Token den Rückgabewert *CKR_RANDOM_SEED_NOT_SUPPORTED* zurück. Dies zeigt an, dass ein Seed nicht verwendet werden kann.
Das Generieren von Zufallswerten geschieht bei der Smartcard nicht auf dem Token selber. Dies zeigt das Flag *CKF_RNG* an. Ist es gesetzt, so werden die Zufallswerte auf dem Token erzeugt. Bei dem IDProtect ist dies der Fall. Die Zeitwerte der Smartcard sagen dementsprechend wenig über das Token selber aus. Tabelle 19 zeigt, wie sich bei fünfmaligem Ausführen von TCH die Zeitwerte der Zufallswertegenerierung auf dem Key verhält. Es ergeben sich die Mittelwerte $8,8843 Sekunden$, $72,8612 Sekunden$ und $580,7852 Sekunden$ mit einer Standardabweichung von $0,0654 Sekunden$, $0,4512 Sekunden$ und $3,9400 Sekunden$ bei einer Erzeugung von je 1000 mal 64, 1024 und 8192 Bytes.

	1 Versuch	2 Versuch	3 Versuch	4 Versuch	5 Versuch
64 Byte	8,966778	8,882520	8,948198	8,807006	8,817106
1024 Byte	73,408349	73,323137	72,639152	72,738442	72,197154
8192 Byte	583,784663	578,592440	586,929826	578,283774	576,335358

Tabelle 19: *Dauer der Generierung von Zufallswerten in Sekunden mit dem IDProtect Key. Jeweils 1000 Funktionsaufrufe pro Versuch.*

7 Fazit

Ziel der Arbeit war es, eine Möglichkeit zu schaffen, kryptographische Hardware zu testen. Für Entscheider und Anwender jener Hardware sollte eine Möglichkeit geschaffen werden, ihre Hardware zu testen und so genaue und verlässliche Informationen zu erhalten.
In Kapitel 2 wurde der Begriff der kryptographischen Hardware geklärt und einige Schnittstellen vorgestellt, mit denen diese für Entwickler zugänglich wird. Im Rahmen dieser Arbeit wurde sich entschieden Cryptoki, die aus PKCS #11 hervorgehenden Schnittstelle, als Grundlage für Tests an kryptographischer Hardware zu verwenden. So wurden in Kapitel 4, die Abhängigkeitsstrukturen zwischen den Funktionen von Cryptoki untersucht und so möglichen Anwendungsfälle aufgedeckt. Darauf basierend wurden 25 Testfälle entworfen, die die Funktionen von Cryptoki zumindest grundlegend testen sollen. Diese wurden mit einem erarbeiteten Ablauf der Software in Abschnitt 4.3 vorgestellt.
Die Implementierung der daraus resultierenden Software, namens TCH, ist in Kapitel 5 dokumentiert.
In Kapitel 6 wurden schließlich zwei verschiedene Token auf Ihren Funktionsumfang im Rahmen der Cryptoki Schnittstelle getestet. Die Ergebnisse zeigen, dass TCH als Resultat des Entwurfes aus Kapitel 4 einige Funktionsumfänge und Besonderheiten offenlegen kann. Es sind noch nicht alle Mechanismen die in PKCS #11 definiert sind unterstützt und können deshalb nicht getestet werden. Ebenso konnten einige Testfälle noch nicht an praktischen Beispielen getestet werden. TCH hat funktionell noch eine Menge Möglichkeiten.
Um einen weiteren Ausblick zu geben, seien hier einige Möglichkeiten erwähnt: Es könnten mehrere Token parallel getestet werden, die Tests flexibler aufgerufen werden, die Informationen, die das Token über sich selber gibt wie z.B. die möglichen PIN-Längen, getestet und weitere Grenzfälle ausprobiert werden.
Wie in Abschnitt 6.2.3 mit den getesteten Token eingetreten, kann der Fall auftreten, dass das Token z.B. nur die Ver- und nicht die Entschlüsselung unterstützt. Oder nur das Nutzen eines Schlüssels aber nicht die Generierung. Diese Fälle werden zurzeit nicht abgedeckt.
Langfristig besteht die Möglichkeit, mit einer Software wie TCH eine Testsuite für kryptographische Hardware zu etablieren. Der Vergleich von kryptographischer Hardware ist aus vielerlei Gründen nicht trivial. Jedoch gibt es mit der Messung der Möglichkeiten in Bezug auf bestimmte Schnittstellen die Möglichkeit, direkte Vergleiche zuzulassen oder zumindest das Sortieren für Anwendungsfälle. Eine solche Testsuite ist

natürlich nicht nur auf Cryptoki begrenzt. Durch das Vorhandensein einer Testsuite, könnten auch Hersteller von kryptographischer Hardware selber Informationen liefern, die über diejenigen aus dem Datenblatt hinausgehen.

Literatur

[1] *An Overview of Hardware Security Modules*. abgerufen am 14.04.2015. URL: http://www.sans.org/reading-room/whitepapers/vpns/overview-hardware-security-modules-757.

[2] *Base Provider Algorithms*. abgerufen am 17.04.2015. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa375599(v=vs.85).aspx.

[3] *Cryptography Essentials*. abgerufen am 07.04.2015. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380251(v=vs.85).aspx.

[4] *CCID*. abgerufen am 13.04.2015. URL: http://www.usb.org/developers/docs/devclass_docs/DWG_Smart-Card_CCID_Rev110.pdf.

[5] *Cryptography API: Next Generation*. abgerufen am 07.05.2015. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa376210(v=vs.85).aspx.

[6] CardLogix Corporation. *Smart Card Standards*. abgerufen am 13.04.2015. URL: http://www.smartcardbasics.com/smart-card-standards.html.

[7] *CSPs and the Cryptography Process*. abgerufen am 07.04.2015. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa381481(v=vs.85).aspx.

[8] *Microsoft Cryptographic Service Providers*. abgerufen am 07.05.2015. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa386983(v=vs.85).aspx.

[9] Claudia Eckert. *IT-Sicherheit*. 978-3-486-72138-6. De Gruyter Oldenbourg, 2013.

[10] *IDPrime .NET*. abgerufen am 29.04.2015. URL: http://www.gemalto.com/Products/dotnet_card/index.html.

[11] *ISO/IEC 7810 Identification cards — Physical characteristics*.

[12] *ISO/IEC 7816-1*. 2011.

[13] *ISO/IEC 7816-15*. 2004.

[14] *ISO/IEC 7816-2:2007, Identification cards – Integrated circuit cards – Part 2: Cards with contacts – Dimensions and location of the contacts*.

[15] NLnet Labs Jelte Jansen. *An introduction to the use of HSM*. NLnet Labs, 13 Mai 2008.

[16] Athena SCS Limited. *PRODUCT DATASHEET IDProtect Key*. URL: http://www.athena-scs.com/docs/products-solutions-datasheets/athena-idprotect-laser.pdf.

[17] Peter Walker. *Bradley Manning trial: what we know from the leaked WikiLeaks documents*. abgerufen am 30.07.2013. URL: http://www.theguardian.com/world/2013/jul/30/bradley-manning-wikileaks-revelations.

[18] *Netscape PKCS #11 Test Suite*. abgerufen am 07.04.2015. URL: http://www-archive.mozilla.org/projects/security/pki/pkcs11/netscape/pk11test.html.

[19] *Netscape PKCS #11 Test Suites*. abgerufen am 07.04.2015. URL: http://www-archive.mozilla.org/projects/security/pki/pkcs11/netscape/.

[20] *Network Security Services*. abgerufen am 07.05.2015. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[21] *OASIS PKCS 11 TC*. abgerufen am 09.05.2015. URL: https://www.oasis-open.org/committees/tc%5C_home.php?wg%5C_abbrev=pkcs11.

[22] *Open Smart Card Development Platform*. abgerufen am 07.04.2015. URL: http://www.openscdp.org/.

[23] *Open source smart card tools and middleware. PKCS#11/MiniDriver/Tokend*. abgerufen am 07.04.2015. URL: https://github.com/OpenSC/OpenSC.

[24] *OpenSC – tools and libraries for smart cards*. abgerufen am 07.04.2015. URL: https://github.com/OpenSC/OpenSC/wiki.

[25] *OpenSCDP Script Collection*. abgerufen am 07.04.2015. URL: http://www.openscdp.org/scripts/index.html.

[26] *p11-tests*. abgerufen am 07.05.2014. URL: http://thewalter.net/git/cgit.cgi/p11-tests.

[27] *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40. Edited by Susan Gleeson and Chris Zimman. 23 December 2014. Candidate OASIS Standard 01*. Dez. 2014.

[28] *PKCS #11 v2.20: Cryptographic Token Interface Standard*. RSA Laboratories, Juni 2004.

[29] *PKCS#11 COVERAGE*. abgerufen am 07.05.2014. URL: http://thewalter.net/git/cgit.cgi/p11-tests/tree/doc/pkcs11-coverage.txt.

[30] RSA Laboratories. *Public-Key Cryptography Standards (PKCS)*. abgerufen am 21.03.2015. URL: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm.

[31] *Private Korrespondenz mit Herr Andreas Lamm vom 16.04.2015. Anhang 2*.

[32] BBC. *Edward Snowden: Leaks that exposed US spy programme*. abgerufen am 17.01.2014. URL: http://www.bbc.com/news/world-us-canada-23123964.

[33] Handelsblatt. *50-Milliarden-Schaden*. abgerufen am 28.08.2013. URL: http://www.handelsblatt.com/politik/deutschland/%5C%5Cwirtschaftsspionage-50-milliarden-schaden/8705934.html.

[34] *Testing third-party PKCS*. abgerufen am 07.04.2015. URL: https://developer.mozilla.org/en-US/docs/Testing_third-party_PKCS#11_modules.2Ftokens_with_NSS.

[35] *PKCS #11 PKCS #6: Extended-Certificate Syntax Standard, Revised November 1, 1993.* abgerufen am 07.04.2015. URL: ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-6.asc.

Anhang

Anhang 1

Konstantenname	Wert
CKR_OK	0*x*00000000
CKR_CANCEL	0*x*00000001
CKR_HOST_MEMORY	0*x*00000002
CKR_SLOT_ID_INVALID	0*x*00000003
CKR_GENERAL_ERROR	0*x*00000005
CKR_FUNCTION_FAILED	0*x*00000006
CKR_ARGUMENTS_BAD	0*x*00000007
CKR_NO_EVENT	0*x*00000008
CKR_NEED_TO_CREATE_THREADS	0*x*00000009
CKR_CANT_LOCK	0*x*0000000*A*
CKR_ATTRIBUTE_READ_ONLY	0*x*00000010
CKR_ATTRIBUTE_SENSITIVE	0*x*00000011
CKR_ATTRIBUTE_TYPE_INVALID	0*x*00000012
CKR_ATTRIBUTE_VALUE_INVALID	0*x*00000013
CKR_DATA_INVALID	0*x*00000020
CKR_DATA_LEN_RANGE	0*x*00000021
CKR_DEVICE_ERROR	0*x*00000030
CKR_DEVICE_MEMORY	0*x*00000031
CKR_DEVICE_REMOVED	0*x*00000032
CKR_ENCRYPTED_DATA_INVALID	0*x*00000040
CKR_ENCRYPTED_DATA_LEN_RANGE	0*x*00000041
CKR_FUNCTION_CANCELED	0*x*00000050
CKR_FUNCTION_NOT_PARALLEL	0*x*00000051
CKR_FUNCTION_NOT_SUPPORTED	0*x*00000054
CKR_KEY_HANDLE_INVALID	0*x*00000060
CKR_KEY_SIZE_RANGE	0*x*00000062
CKR_KEY_TYPE_INCONSISTENT	0*x*00000063
CKR_KEY_NOT_NEEDED	0*x*00000064
CKR_KEY_CHANGED	0*x*00000065

Tabelle 20: *PKCS #11 Rückgabewerte mit Konstantennamen [28]*

Konstantenname	Wert
CKR_KEY_NEEDED	0x00000066
CKR_KEY_INDIGESTIBLE	0x00000067
CKR_KEY_FUNCTION_NOT_PERMITTED	0x00000068
CKR_KEY_NOT_WRAPPABLE	0x00000069
CKR_KEY_UNEXTRACTABLE	0x0000006A
CKR_MECHANISM_INVALID	0x00000070
CKR_MECHANISM_PARAM_INVALID	0x00000071
CKR_OBJECT_HANDLE_INVALID	0x00000082
CKR_OPERATION_ACTIVE	0x00000090
CKR_OPERATION_NOT_INITIALIZED	0x00000091
CKR_PIN_INCORRECT	0x000000A0
CKR_PIN_INVALID	0x000000A1
CKR_PIN_LEN_RANGE	0x000000A2
CKR_PIN_EXPIRED	0x000000A3
CKR_PIN_LOCKED	0x000000A4
CKR_SESSION_CLOSED	0x000000B0
CKR_SESSION_COUNT	0x000000B1
CKR_SESSION_HANDLE_INVALID	0x000000B3
CKR_SESSION_PARALLEL_NOT_SUPPORTED	0x000000B4
CKR_SESSION_READ_ONLY	0x000000B5
CKR_SESSION_EXISTS	0x000000B6
CKR_SESSION_READ_ONLY_EXISTS	0x000000B7
CKR_SESSION_READ_WRITE_SO_EXISTS	0x000000B8
CKR_SIGNATURE_INVALID	0x000000C0
CKR_SIGNATURE_LEN_RANGE	0x000000C1
CKR_TEMPLATE_INCOMPLETE	0x000000D0
CKR_TEMPLATE_INCONSISTENT	0x000000D1
CKR_TOKEN_NOT_PRESENT	0x000000E0
CKR_TOKEN_NOT_RECOGNIZED	0x000000E1
CKR_TOKEN_WRITE_PROTECTED	0x000000E2
CKR_UNWRAPPING_KEY_HANDLE_INVALID	0x000000F0
CKR_UNWRAPPING_KEY_SIZE_RANGE	0x000000F1
CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT	0x000000F2
CKR_USER_ALREADY_LOGGED_IN	0x00000100
CKR_USER_NOT_LOGGED_IN	0x00000101
CKR_USER_PIN_NOT_INITIALIZED	0x00000102
CKR_USER_TYPE_INVALID	0x00000103
CKR_USER_ANOTHER_ALREADY_LOGGED_IN	0x00000104
CKR_USER_TOO_MANY_TYPES	0x00000105
CKR_WRAPPED_KEY_INVALID	0x00000110
CKR_WRAPPED_KEY_LEN_RANGE	0x00000112

Tabelle 20: *PKCS #11 Rückgabewerte mit Konstantennamen [28]*

Konstantenname	Wert
CKR_WRAPPING_KEY_HANDLE_INVALID	0*x*00000113
CKR_WRAPPING_KEY_SIZE_RANGE	0*x*00000114
CKR_WRAPPING_KEY_TYPE_INCONSISTENT	0*x*00000115
CKR_RANDOM_SEED_NOT_SUPPORTED	0*x*00000120
CKR_RANDOM_NO_RNG	0*x*00000121
CKR_DOMAIN_PARAMS_INVALID	0*x*00000130
CKR_BUFFER_TOO_SMALL	0*x*00000150
CKR_SAVED_STATE_INVALID	0*x*00000160
CKR_INFORMATION_SENSITIVE	0*x*00000170
CKR_STATE_UNSAVEABLE	0*x*00000180
CKR_CRYPTOKI_NOT_INITIALIZED	0*x*00000190
CKR_CRYPTOKI_ALREADY_INITIALIZED	0*x*00000191
CKR_MUTEX_BAD	0*x*000001*A*0
CKR_MUTEX_NOT_LOCKED	0*x*000001*A*1
CKR_NEW_PIN_MODE	0*x*000001*B*0
CKR_NEXT_OTP	0*x*000001*B*1
CKR_FUNCTION_REJECTED	0*x*00000200
CKR_VENDOR_DEFINED	0*x*80000000

Tabelle 20: *PKCS #11 Rückgabewerte mit Konstantennamen [28]*

Anhang 2

On 16.04.2015 12:51, Lamm, Andreas wrote:
> > Lieber Herr Häring,
> >
> > die Position der Kontakte ist in ISO/IEC 7816-2 festgelegt. Ich habe Ihnen dazu das Bild aus der Norm mit den Maßen angehangen.
> > Die Lage des Chips /des Siliziums oder auch des Chipmoduls ist im Grunde nicht festgelegt, sondern nur wo die o.a. Aussenkontakte liegen müssen.
> > Rein theoretisch könnten Chip/Chipmodul ganz woanders in der Karte "versteckt" sein, mit einer Verdrahtung hin zu den ISO Kontakten. So eine Konstruktion gibt es nach Meinung unserer Experten aber nicht.
> > Die tatsächliche Dimension der Chipmodule ist nicht festgelegt.
> >
> > Die Ecken müssen nach ISO/IEC 7810 mit einem festgelegten Radius abgerundet werden. Auch dazu habe ich Ihnen ein Bild angehangen, in dem Sie die Festlegungen für ID-1 finden.
> >

> > Ich hoffe diese Information helfen Ihnen bei der Erstellung des Schaubildes.
> >
> > Mit freundlichen Grüßen.
> >
> > DIN Deutsches Institut für Normung e. V.
> > Normenausschuss Informationstechnik und Anwendungen (NIA)
> >
> > Andreas Lamm
> > Projektmanager
> >
> > Telefon: +49 30 2601-2064
> > Fax: +49 30 2601 42064
> > E-Mail: andreas.lamm@din.de
> > Internet: http://*www.din.de*
> > Internet: http://*www.nia.din.de*
> >
> > DIN Deutsches Institut für Normung e. V.
> > Sitz: Am DIN-Platz, Burggrafenstraße 6, 10787 Berlin
> > Präsident: Prof. Dr.-Ing. Klaus Homann
> > Vorstand: Dr.-Ing. Torsten Bahke (Vorsitzender), Dipl.-Ing. Rüdiger Marquardt
> > Geschäftsleitung: Dr.-Ing. Ulrike Bohnsack, Dr.-Ing. Karlhanns Gindele, Dipl.-Kfm. Daniel Schmidt,
> > Dr. rer. nat. Hartmut Strauß, Astrid Wirges, Dipl.-Ing. Ernst-Peter Ziethen
> > Registergericht: AG Berlin-Charlottenburg, VR 288 B
> >
> > Mitglied der Internationalen Organisation für Normung (ISO) und des Europäischen Komitees für Normung (CEN)
> >
> > Zertifiziert nach DIN EN ISO 9001:2008-12
> >
> > Der Inhalt dieser E-Mail (einschließlich etwaiger beigefü gter Dateien) ist vertraulich
> > und nur für den Empfänger bestimmt. Sollten Sie nicht der bestimmungsgemäße Empfänger sein, ist Ihnen jegliche Offenlegung, Vervielfältigung, Weitergabe oder Nutzung des Inhalts untersagt. Bitte informieren Sie in diesem Fall unverzüglich den Absender und löschen Sie die E-Mail (einschließlich etwaiger beigefügter Dateien) von Ihrem System.
> > Vielen Dank.

Anhang 3

```
Testing implementation of functions ...
 C_Initialize: Supported.
 C_GetInfo: Supported.
 C_GetSlotList: Supported.
 C_GetSlotInfo: Supported.
 C_GetTokenInfo: Supported.
 C_WaitForSlotEvent(CKF_DONT_BLOCK): Supported.
 C_GetMechanismList: Supported.
 C_GetMechanismInfo: Supported.
 C_InitToken: Supported.
 C_InitPIN: Supported.
 C_SetPIN: Supported.
 C_OpenSession: Supported.
 C_CloseSession: Supported.
 C_CloseAllSessions: Supported.
 C_GetSessionInfo: Supported.
 C_GetOperationState: Not supported.
 C_SetOperationState: Not supported.
 C_Login: Supported.
 C_Logout: Supported.
 C_CreateObject: Supported.
 C_CopyObject: Not supported.
 C_DestroyObject: Supported.
 C_GetObjectSize: Supported.
 C_GetAttributeValue: Supported.
 C_SetAttributeValue: Supported.
 C_FindObjectsInit: Supported.
 C_FindObjects: Supported.
 C_FindObjectsFinal: Supported.
 C_EncryptInit: Supported.
 C_Encrypt: Supported.
 C_EncryptUpdate: Supported.
 C_EncryptFinal: Supported.
 C_DecryptInit: Supported.
 C_Decrypt: Supported.
 C_DecryptUpdate: Supported.
 C_DecryptFinal: Supported.
 C_DigestInit: Supported.
 C_Digest: Supported.
 C_DigestUpdate: Supported.
 C_DigestKey: Supported.
 C_DigestFinal: Supported.
 C_SignInit: Supported.
```

```
 C_Sign: Supported.
 C_SignUpdate: Supported.
 C_SignFinal: Supported.
 C_SignRecoverInit: Supported.
 C_SignRecover: Supported.
 C_VerifyInit: Supported.
 C_Verify: Supported.
 C_VerifyUpdate: Supported.
 C_VerifyFinal: Supported.
 C_VerifyRecoverInit: Supported.
 C_VerifyRecover: Supported.
 C_DigestEncryptUpdate: Not supported.
 C_DecryptDigestUpdate: Not supported.
 C_SignEncryptUpdate: Not supported.
 C_DecryptVerifyUpdate: Not supported.
 C_GenerateKey: Supported.
 C_GenerateKeyPair: Supported.
 C_WrapKey: Not supported.
 C_UnwrapKey: Supported.
 C_DeriveKey: Supported.
 C_SeedRandom: Supported.
 C_GenerateRandom: Supported.
 C_Finalize: Supported.
C_GetInfo ...
 Cryptoki Version: 2.20
 Manufacturer ID: Athena Smartcard Solutions
 Flags: 0 (should be null)
 Library Description: ASE Cryptoki
 Library Version: 3.1
C_GetSlotInfo on slot 0
 Slot Description: Athena IDProtect Key v2 [Main Interface] 00
    00
 Manufacturer ID:
 Flags(7): CKF_TOKEN_PRESENT, CKF_REMOVABLE_DEVICE, CKF_HW_SLOT
    ,
 Hardware Version: 1.0
 Firmware Version: 1.0
C_GetTokenInfo on slot 0
 Label: IDProtect
 Manufacturer ID: Athena Smartcard Solutions
 Model: IDProtect
 Serialnumber: 1111111111111111
 Flags (4195341): CKF_RNG, CKF_LOGIN_REQUIRED,
    CKF_USER_PIN_INITIALIZED, CKF_TOKEN_INITIALIZED,
    CKF_SO_PIN_LOCKED,
```

```
 Max sessions: 1000
 Open sessions: 0
 Max R/W sessions: 1000
 Open R/W session: 0
 Max. PIN length: 16
 Min. PIN length: 4
 Public memory: 18446744073709551615
 Free public memory: 50092
 Private memory: 18446744073709551615
 Free private memory: 50092
 Hardware Version: 1.0
 Firmware Version: 1.0
 Time:
C_GetMechanismList on slot 0
CKM_RSA_PKCS, CKM_RSA_X_509, CKM_SHA1_RSA_PKCS,
    CKM_RSA_PKCS_OAEP, CKM_RSA_PKCS_PSS, CKM_SHA256_RSA_PKCS,
    CKM_SHA384_RSA_PKCS, CKM_SHA512_RSA_PKCS, CKM_MD5_RSA_PKCS,
    CKM_RSA_PKCS_KEY_PAIR_GEN, CKM_DES_KEY_GEN, CKM_DES2_KEY_GEN
    , CKM_DES3_KEY_GEN, CKM_DES_ECB, CKM_DES3_ECB, CKM_DES_CBC,
    CKM_DES_MAC_GENERAL, CKM_DES_MAC, CKM_DES3_CBC,
    CKM_DES3_MAC_GENERAL, CKM_DES3_MAC, CKM_SHA_1,
    CKM_SHA_1_HMAC, CKM_MD5, CKM_MD5_HMAC, CKM_AES_KEY_GEN,
    CKM_AES_ECB, CKM_AES_CBC, CKM_AES_MAC, CKM_AES_MAC_GENERAL,
    CKM_SHA256, CKM_SHA256_HMAC, CKM_SHA384, CKM_SHA384_HMAC,
    CKM_SHA512, CKM_SHA512_HMAC,
C_GetMechanismInfo on slot 0
 CKM_RSA_PKCS_KEY_PAIR_GEN: key size(1024-2048)CKF_HW,
    CKF_GENERATE_KEY_PAIR,
 CKM_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_ENCRYPT,
    CKF_DECRYPT, CKF_SIGN, CKF_VERIFY,
 CKM_RSA_X_509: key size(1024-2048)CKF_HW, CKF_ENCRYPT,
    CKF_DECRYPT, CKF_VERIFY,
 CKM_MD5_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_SHA1_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_RSA_PKCS_OAEP: key size(1024-2048)CKF_ENCRYPT, CKF_DECRYPT
    , CKF_UNWRAP,
 CKM_RSA_PKCS_PSS: key size(1024-2048)CKF_SIGN, CKF_VERIFY,
 CKM_SHA256_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_SHA384_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_SHA512_RSA_PKCS: key size(1024-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
```

```
 CKM_DES_KEY_GEN: key size(8-8)CKF_GENERATE,
 CKM_DES_ECB: key size(8-8)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
    CKF_UNWRAP,
 CKM_DES_CBC: key size(8-8)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
    CKF_UNWRAP,
 CKM_DES_MAC: key size(8-24)CKF_HW, CKF_SIGN, CKF_VERIFY,
 CKM_DES_MAC_GENERAL: key size(8-24)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_DES2_KEY_GEN: key size(16-16)CKF_GENERATE,
 CKM_DES3_KEY_GEN: key size(24-24)CKF_GENERATE,
 CKM_DES3_ECB: key size(16-24)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
     CKF_UNWRAP,
 CKM_DES3_CBC: key size(16-24)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
     CKF_UNWRAP,
 CKM_DES3_MAC: key size(16-24)CKF_HW, CKF_SIGN, CKF_VERIFY,
 CKM_DES3_MAC_GENERAL: key size(16-24)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_MD5: key size(0-0)CKF_DIGEST,
 CKM_MD5_HMAC: key size(8-24)CKF_SIGN, CKF_VERIFY,
 CKM_SHA_1: key size(0-0)CKF_DIGEST,
 CKM_SHA_1_HMAC: key size(8-24)CKF_SIGN, CKF_VERIFY,
 CKM_SHA256: key size(0-0)CKF_DIGEST,
 CKM_SHA256_HMAC: key size(8-24)CKF_SIGN, CKF_VERIFY,
 CKM_SHA384: key size(0-0)CKF_DIGEST,
 CKM_SHA384_HMAC: key size(8-24)CKF_SIGN, CKF_VERIFY,
 CKM_SHA512: key size(0-0)CKF_DIGEST,
 CKM_SHA512_HMAC: key size(8-24)CKF_SIGN, CKF_VERIFY,
 CKM_AES_KEY_GEN: key size(16-32)CKF_GENERATE,
 CKM_AES_ECB: key size(16-32)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
    CKF_UNWRAP,
 CKM_AES_CBC: key size(16-32)CKF_HW, CKF_ENCRYPT, CKF_DECRYPT,
    CKF_UNWRAP,
 CKM_AES_MAC: key size(16-32)CKF_HW, CKF_SIGN, CKF_VERIFY,
 CKM_AES_MAC_GENERAL: key size(16-32)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
Find objects on slot 0 as User
 Could find 0 objects of class 0
 Could find 0 objects of class 1
 Could find 0 objects of class 3
 Could find 0 objects of class 8
 Could find 0 objects of class 2
 Could find 0 objects of class 4
 Could find 0 objects of class 2147483648
Find public objects on slot 0
 Could find 0 objects of class 0
```

```
 Could find 0 objects of class 1
 Could find 0 objects of class 3
 Could find 0 objects of class 8
 Could find 0 objects of class 2
 Could find 0 objects of class 4
 Could find 0 objects of class 2147483648
Testing case: Minimal ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call GetInfo ... success
 call Finalize ... success
 ... case successful
Testing case: SlotInformation ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call GetSlotList with CK_FALSE and NULL_PTR ... success
 call GetSlotList with CK_FALSE ... success
 call GetSlotInfo on slot 0 ... success
 call Finalize ... success
 ... case successful
Testing case: TokenInformation ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_GetTokenInfo on slot 0 ... success
 call GetMechanismList with NULL_PTR on slot 0 ... success
 call GetMechanismList on slot 0 ... success
 call Finalize ... success
 ... case successful
Testing case: CompareMechanismLists ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call GetMechanismList with NULL_PTR on slot 0 ... success
 call GetMechanismList on slot 0 ... success
 iterate over all mechanisms and call C_GetMechanismInfo on
    slot 0 ... compared with MechanismList and the result is
    off by 0 mechanisms
```

```
 iterate over mechanismlist(C_GetMechanismList) and call
    C_GetMechanismInfo
 call Finalize ... success
 ... case successful
Testing case: SingleSessionHandling ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION... failed as
    expected
 call C_GetSessionInfo on slot 0 with the previous session
    handle... failed as expected
 call C_CloseSession on slot 0 with the previous session handle
    ... failed as expected
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after logout)
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(session handle is invalid after close)
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after logout)
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(session handle is invalid after close)
 call C_Finalize ... success
```

```
 ... case successful
Testing case: MultipleSessionHandling ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after login)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after logout)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after logout)
 call C_CloseAllSessions on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(session handle is invalid after close)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(session handle is invalid after close)
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after login)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after logout)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after logout)
 call C_CloseAllSessions on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(session handle is invalid after close)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(session handle is invalid after close)
```

```
 call C_Finalize ... success
 ... case successful
Testing case: PreserveOperationState ...
 Check dependencies ...
  MISSING: C_GetOperationState
Testing case: AdminToken ...
 Check dependencies ...
 expected argument '-i' was not ommited
Testing case: ChangePin ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_SetPIN on slot 0 as User ...C_SetPIN: CKR_PIN_INVALID
: CKR_PIN_INVALID
 call C_Logout on slot 0 ... success
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_Finalize ... success
 ... case successful
Testing case: ObjectHandling ...
 Check dependencies ...
  MISSING: _DestroyObject
Testing case: ObjectSearch ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_CreateObject with object class 0... success
 call C_CreateObject with object class 0... success
 call C_CreateObject with object class 0... success
 could create 3 data objects. So we should find 3 more than at
    the start
 Could find 3 objects of class 0
 call C_Logout on slot 0 ... success
 call C_CloseSession on slot 0... success
 call Finalize ... success
```

```
 ... case successful
Testing case: DeAndEncryptionSimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 De - and encrypting(CKM_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
   Could encrypt 16 CK_BYTEs in 0.025862 sec.
   Could decrypt 96 CK_BYTEs in 0.970410 sec.
C_Encrypt: CKR_DATA_LEN_RANGE
    failed
C_Encrypt: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_RSA_X_509) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
   Could encrypt 16 CK_BYTEs in 0.015864 sec.
   Could decrypt 96 CK_BYTEs in 0.952439 sec.
C_Encrypt: CKR_DATA_LEN_RANGE
    failed
C_Encrypt: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_RSA_PKCS_OAEP) ...
  skip DeAndEncrypting(CKM_RSA_PKCS_OAEP) because the related
     key generation mechanism was not found...
 De - and encrypting(CKM_DES_ECB) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
   Could encrypt 16 CK_BYTEs in 0.001460 sec.
   Could decrypt 16 CK_BYTEs in 0.001341 sec.
```

```
  Could encrypt 1048576 CK_BYTEs in 0.264360 sec.
  Could decrypt 1048576 CK_BYTEs in 0.265335 sec.
  Could encrypt 33554432 CK_BYTEs in 8.435352 sec.
  Could decrypt 33554432 CK_BYTEs in 8.454801 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
De - and encrypting(CKM_DES3_ECB) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
 encrypt + decrypt ...
  Could encrypt 16 CK_BYTEs in 0.001659 sec.
  Could decrypt 16 CK_BYTEs in 0.001677 sec.
  Could encrypt 1048576 CK_BYTEs in 0.791334 sec.
  Could decrypt 1048576 CK_BYTEs in 0.791547 sec.
  Could encrypt 33554432 CK_BYTEs in 25.260214 sec.
  Could decrypt 33554432 CK_BYTEs in 25.268579 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
De - and encrypting(CKM_DES_CBC) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
 encrypt + decrypt ...
  Could encrypt 16 CK_BYTEs in 0.000351 sec.
  Could decrypt 8 CK_BYTEs in 0.000339 sec.
  Could encrypt 1048576 CK_BYTEs in 0.272305 sec.
  Could decrypt 1048568 CK_BYTEs in 0.272669 sec.
  Could encrypt 33554432 CK_BYTEs in 8.693740 sec.
  Could decrypt 33554424 CK_BYTEs in 8.676414 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
De - and encrypting(CKM_DES3_CBC) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
 encrypt + decrypt ...
  Could encrypt 16 CK_BYTEs in 0.001656 sec.
  Could decrypt 8 CK_BYTEs in 0.001532 sec.
  Could encrypt 1048576 CK_BYTEs in 0.793642 sec.
  Could decrypt 1048568 CK_BYTEs in 0.794501 sec.
  Could encrypt 33554432 CK_BYTEs in 25.478435 sec.
```

```
    Could decrypt 33554424 CK_BYTEs in 25.450130 sec.
  call C_Logout ... success
  call C_CloseSession with the previous session handle...
     success
 De - and encrypting(CKM_AES_ECB) ...
  call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
     CKF_RW_SESSION... success
  call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
   skip DeAndEncrypting(CKM_AES_ECB) because mecha is not yet
      implemented ...
  call C_Logout ... success
  call C_CloseSession with the previous session handle...
     success
 De - and encrypting(CKM_AES_CBC) ...
  call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
     CKF_RW_SESSION... success
  call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
   skip DeAndEncrypting(CKM_AES_CBC) because mecha is not yet
      implemented ...
  call C_Logout ... success
  call C_CloseSession with the previous session handle...
     success
  call Finalize ... success
  ... case successful
Testing case: DeAndEncryptionMultiple ...
 Check dependencies ...
 Dependencies are fine
  call Initialize ... success
  call C_GetSlotList with CK_TRUE and NULL_PTR ... success
  call C_GetSlotList with CK_TRUE ... success
 De - and encrypting(CKM_RSA_PKCS) ...
  call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
     CKF_RW_SESSION... success
  call C_Login as User ... success
   encrypt + decrypt ...
C_EncryptFinal: CKR_DATA_LEN_RANGE
     failed
  call C_Logout ... success
  call C_CloseSession with the previous session handle...
     success
 De - and encrypting(CKM_RSA_X_509) ...
  call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
     CKF_RW_SESSION... success
```

```
 call C_Login as User ... success
  encrypt + decrypt ...
C_EncryptFinal: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_RSA_PKCS_OAEP) ...
  skip DeAndEncrypting(CKM_RSA_PKCS_OAEP) because related key
     generation mecha was not found ...
 De - and encrypting(CKM_DES_ECB) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
C_EncryptUpdate NULL_PTR: CKR_BUFFER_TOO_SMALL
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_DES3_ECB) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
C_EncryptUpdate NULL_PTR: CKR_BUFFER_TOO_SMALL
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_DES_CBC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
C_EncryptUpdate: CKR_BUFFER_TOO_SMALL
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_DES3_CBC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
```

```
C_EncryptUpdate: CKR_BUFFER_TOO_SMALL
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_AES_ECB) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip DeAndEncrypting(CKM_AES_ECB) because mecha is not yet
     implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_AES_CBC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip DeAndEncrypting(CKM_AES_CBC) because mecha is not yet
     implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call C_Logout ...: CKR_SESSION_HANDLE_INVALID
 call Finalize ... success
 ... case successful
Testing case: DigestSimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 Digest:CKM_SHA_1 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
2024f600b51d0aa19daa1729a85ceb1e9ef5864f <-
   xvwvhwunygbltpezvpykkcuwtdjdsxxsuvpbulpsrqflhjmeakonpkjinslirkcl

   Try to digest a Key from CKM_DES_KEY_GEN
 call C_Login as User ... success
```

```
  call C_DigestInit on slot 0 ... success
  call C_DigestKey on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
 call C_Logout ... success
 Digest:CKM_MD5 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
c0aa8705efcc0d3b8e963ad1655cbb37 <-
   xvwvhwunygbltpezvpykkcuwtdjdsxxsuvpbulpsrqflhjmeakonpkjinslirkcl

   Try to digest a Key from CKM_DES_KEY_GEN
 call C_Login as User ... success
  call C_DigestInit on slot 0 ... success
  call C_DigestKey on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
 call C_Logout ... success
 Digest:CKM_SHA256 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
bdc820f83767321df569ff2fe57091d1b0c7359bb8e6d4fb25fd917be03c2483
    <-
   xvwvhwunygbltpezvpykkcuwtdjdsxxsuvpbulpsrqflhjmeakonpkjinslirkcl

   Try to digest a Key from CKM_DES_KEY_GEN
 call C_Login as User ... success
  call C_DigestInit on slot 0 ... success
  call C_DigestKey on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
 call C_Logout ... success
 Digest:CKM_SHA384 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
45e803f451e5e6532b48ae08aca01b15c904a30aa938f28d
 1768eed6c7ababc07884ce9e4b8ac382abe080c643f49a11 <-
    xvwvhwunygbltpezvpykkcuwtdjdsxxsuvpbulpsrqflhjmeakonpkjinslirkcl

   Try to digest a Key from CKM_DES_KEY_GEN
 call C_Login as User ... success
  call C_DigestInit on slot 0 ... success
```

```
  call C_DigestKey on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
 call C_Logout ... success
 Digest:CKM_SHA512 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
416f0ad54170b009415555c1f96717b57dd9e2435f095425851
542693f222947d2456afffc4ae1becd12430a75a7f65c70
376b209c4185c4b4682e833073b14e <-
   xvwvhwunygbltpezvpykkcuwtdjdsxxsuvpbulpsrqflhjmeakonpkjinslirkcl

   Try to digest a Key from CKM_DES_KEY_GEN
 call C_Login as User ... success
  call C_DigestInit on slot 0 ... success
  call C_DigestKey on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: DigestMultiple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 Digest:CKM_SHA_1 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
2131fd521e0988c7d203af2eed6256ae55e718c4 <-
   epljmjegznhcewhdtfivnvnzkpxltbazrnkfwolwdvairinmpyjftxefodqhfsiy

 Digest:CKM_MD5 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
```

```
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
b8f7d0ad658da1db3e20899d9663ec4d <-
   epljmjegznhcewhdtfivnvnzkpxltbazrnkfwolwdvairinmpyjftxefodqhfsiy

 Digest:CKM_SHA256 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
ab6613292078d2d379f2918dad14a59dfcaf098f9073a17854c67124bcc9b2d3
    <-
   epljmjegznhcewhdtfivnvnzkpxltbazrnkfwolwdvairinmpyjftxefodqhfsiy

 Digest:CKM_SHA384 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
97998c95ff1e28dc6d68d91835db36afbd32366dbec2
 63ed3be6c24521c164e0d2ad5b7207a801edc9ba0beb7a00f994 <-
    epljmjegznhcewhdtfivnvnzkpxltbazrnkfwolwdvairinmpyjftxefodqhfsiy

 Digest:CKM_SHA512 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
cbdde1757d42051640044a9f36fbd003dobee7578316d2397d282815
d9b4634e9a4e84b705f7cff9b3e05c8e74530592ce0b6269437bffd
39911131caf9fd2a7a <-
   epljmjegznhcewhdtfivnvnzkpxltbazrnkfwolwdvairinmpyjftxefodqhfsiy

 call Finalize ... success
 ... case successful
Testing case: SignAndVerifySimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 Signing and Verify(CKM_RSA_PKCS) ...
```

```
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.792152 sec.
   Could verify 96 CK_BYTEs in 0.015092 sec.
C_Sign: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_RSA_X_509) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.795577 sec.
   Could verify 96 CK_BYTEs in 0.012571 sec.
C_Sign: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA1_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.780976 sec.
   Could verify 96 CK_BYTEs in 0.014837 sec.
   Could sign 256 CK_BYTEs in 0.279001 sec.
   Could verify 96 CK_BYTEs in 0.012935 sec.
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_RSA_PKCS_PSS) ...
  skip CKM_RSA_PKCS_PSS because the related key generation
     mechanism was not found...
 Signing and Verify(CKM_SHA256_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.779739 sec.
   Could verify 96 CK_BYTEs in 0.025136 sec.
   Could sign 256 CK_BYTEs in 0.293856 sec.
   Could verify 96 CK_BYTEs in 0.009704 sec.
 call C_Logout ... success
```

```
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA384_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.776126 sec.
   Could verify 96 CK_BYTEs in 0.015505 sec.
   Could sign 256 CK_BYTEs in 0.283021 sec.
   Could verify 96 CK_BYTEs in 0.016617 sec.
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA512_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.780807 sec.
   Could verify 96 CK_BYTEs in 0.013444 sec.
   Could sign 256 CK_BYTEs in 0.277718 sec.
   Could verify 96 CK_BYTEs in 0.013531 sec.
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_MD5_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.786061 sec.
   Could verify 96 CK_BYTEs in 0.010277 sec.
   Could sign 256 CK_BYTEs in 0.291892 sec.
   Could verify 96 CK_BYTEs in 0.017889 sec.
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES_MAC_GENERAL) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000240 sec.
C_Verify: CKR_ARGUMENTS_BAD
    failed
   Could sign 256 CK_BYTEs in 0.000218 sec.
C_VerifyInit: CKR_OPERATION_ACTIVE
    failed
```

```
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000867 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
   Could sign 256 CK_BYTEs in 0.000947 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES3_MAC_GENERAL) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000988 sec.
C_Verify: CKR_ARGUMENTS_BAD
    failed
   Could sign 256 CK_BYTEs in 0.001036 sec.
C_VerifyInit: CKR_OPERATION_ACTIVE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES3_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000973 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
   Could sign 256 CK_BYTEs in 0.001011 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA_1_HMAC) ...
  skip CKM_SHA_1_HMAC because the related key generation
     mechanism was not found...
```

```
 Signing and Verify(CKM_MD5_HMAC) ...
  skip CKM_MD5_HMAC because the related key generation
      mechanism was not found...
 Signing and Verify(CKM_AES_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_AES_MAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_AES_MAC_GENERAL) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_AES_MAC_GENERAL because mecha is not yet implemented
      ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA256_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_SHA256_HMAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA384_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_SHA384_HMAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA512_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
```

```
 skip CKM_SHA512_HMAC because mecha is not yet implemented ...
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
call Finalize ... success
... case successful
Testing case: SignAndVerifyMultiple ...
Check dependencies ...
Dependencies are fine
call Initialize ... success
call C_GetSlotList with CK_TRUE and NULL_PTR ... success
call C_GetSlotList with CK_TRUE ... success
Signing and Verify(CKM_RSA_PKCS) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
 action is not supported by the key
   failed
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_RSA_X_509) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
 action is not supported by the key
   failed
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_SHA1_RSA_PKCS) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
  Could sign 16 CK_BYTEs in 0.796007 sec.
  Could verify 96 CK_BYTEs in 0.020888 sec.
  Could sign 256 CK_BYTEs in 0.290205 sec.
  Could verify 96 CK_BYTEs in 0.015010 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_RSA_PKCS_PSS) ...
 skip CKM_RSA_PKCS_PSS because the related key generation
    mechanism was not found...
Signing and Verify(CKM_SHA256_RSA_PKCS) ...
```

```
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
  Could sign 16 CK_BYTEs in 0.782879 sec.
  Could verify 96 CK_BYTEs in 0.018545 sec.
  Could sign 256 CK_BYTEs in 0.274003 sec.
  Could verify 96 CK_BYTEs in 0.005789 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_SHA384_RSA_PKCS) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
  Could sign 16 CK_BYTEs in 0.782291 sec.
  Could verify 96 CK_BYTEs in 0.009316 sec.
  Could sign 256 CK_BYTEs in 0.295002 sec.
  Could verify 96 CK_BYTEs in 0.015183 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_SHA512_RSA_PKCS) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
  Could sign 16 CK_BYTEs in 0.792105 sec.
  Could verify 96 CK_BYTEs in 0.018657 sec.
  Could sign 256 CK_BYTEs in 0.281776 sec.
  Could verify 96 CK_BYTEs in 0.012798 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_MD5_RSA_PKCS) ...
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
   CKF_RW_SESSION... success
call C_Login as User ... success
  Could sign 16 CK_BYTEs in 0.782346 sec.
  Could verify 96 CK_BYTEs in 0.023674 sec.
  Could sign 256 CK_BYTEs in 0.283313 sec.
  Could verify 96 CK_BYTEs in 0.009528 sec.
call C_Logout ... success
call C_CloseSession with the previous session handle...
   success
Signing and Verify(CKM_DES_MAC_GENERAL) ...
```

```
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000883 sec.
C_Verify: CKR_ARGUMENTS_BAD
    failed
   Could sign 256 CK_BYTEs in 0.000915 sec.
C_VerifyInit: CKR_OPERATION_ACTIVE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000893 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
   Could sign 256 CK_BYTEs in 0.000946 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES3_MAC_GENERAL) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000263 sec.
C_Verify: CKR_ARGUMENTS_BAD
    failed
   Could sign 256 CK_BYTEs in 0.000242 sec.
C_VerifyInit: CKR_OPERATION_ACTIVE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_DES3_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.000249 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
```

```
   Could sign 256 CK_BYTEs in 0.000616 sec.
C_Verify: CKR_SIGNATURE_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA_1_HMAC) ...
  skip CKM_SHA_1_HMAC because the related key generation
     mechanism was not found...
 Signing and Verify(CKM_MD5_HMAC) ...
  skip CKM_MD5_HMAC because the related key generation
     mechanism was not found...
 Signing and Verify(CKM_AES_MAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_AES_MAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_AES_MAC_GENERAL) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_AES_MAC_GENERAL because mecha is not yet implemented
      ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA256_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_SHA256_HMAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA384_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
```

```
  skip CKM_SHA384_HMAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA512_HMAC) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
C_GenerateKey: CKR_ATTRIBUTE_VALUE_INVALID
  skip CKM_SHA512_HMAC because mecha is not yet implemented ...
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: SignAndVerifyRecover ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call Finalize ... success
 ... case successful
Testing case: DualPurposeFunctions ...
 Check dependencies ...
  MISSING: C_DigestEncryptUpdate
Testing case: SingleKeyGeneration ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_Login on slot 0 as User ... success
 call C_GenerateKey(CKM_DES_KEY_GEN) on slot 0 ... success
 call C_GenerateKey(CKM_DES2_KEY_GEN) on slot 0 ... success
 call C_GenerateKey(CKM_DES3_KEY_GEN) on slot 0 ... success
 call C_GenerateKey(CKM_AES_KEY_GEN) on slot 0 ...C_GenerateKey
    : CKR_ATTRIBUTE_VALUE_INVALID
 call C_Logout on slot 0 ... success
 call C_CloseSession on slot 0... success
 call Finalize ... success
 ... case successful
Testing case: KeyPairGeneration ...
```

```
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_GenerateKeyPair(CKM_RSA_PKCS_KEY_PAIR_GEN) on slot 0
    ... success
 call C_Logout on slot 0 ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: WrapAndUnwrapKey ...
 Check dependencies ...
  MISSING: C_WrapKey
Testing case: DeriveKey ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSIONN...
    success
   testCase is not yet completly implemented. call
      C_CloseSession with the previous session handle...
      success
 call Finalize ... success
 ... case successful
Testing case: GenerateRandom ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call GetSlotInfo on slot 0 ... success
CKF_RNG is set
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_SeedRandom on slot 0 ...: CKR_RANDOM_SEED_NOT_SUPPORTED
 call C_GenerateRandom(64 Bytes) 1000 times on slot 0 ...
  Could call C_GenerateRandom 1000 times with 64 bytes in
     9.035519 sec.
```

```
 append data to file rand_64.txt.
done.
call C_GenerateRandom(1024 Bytes) 1000 times on slot 0 ...
 Could call C_GenerateRandom 1000 times with 1024 bytes in
    74.106388 sec.
 append data to file rand_1024.txt.
done.
call C_GenerateRandom(8192 Bytes) 1000 times on slot 0 ...
 Could call C_GenerateRandom 1000 times with 8192 bytes in
    590.135075 sec.
 append data to file rand_8192.txt.
done.
call Finalize ... success
... case successful
```

Anhang 4

```
Testing implementation of functions ...
 C_Initialize: Supported.
 C_GetInfo: Supported.
 C_GetSlotList: Supported.
 C_GetSlotInfo: Supported.
 C_GetTokenInfo: Supported.
 C_WaitForSlotEvent(CKF_DONT_BLOCK): Supported.
 C_GetMechanismList: Supported.
 C_GetMechanismInfo: Supported.
 C_InitToken: Supported.
 C_InitPIN: Supported.
 C_SetPIN: Supported.
 C_OpenSession: Supported.
 C_CloseSession: Supported.
 C_CloseAllSessions: Supported.
 C_GetSessionInfo: Supported.
 C_GetOperationState: Not supported.
 C_SetOperationState: Not supported.
 C_Login: Supported.
 C_Logout: Supported.
 C_CreateObject: Supported.
 C_CopyObject: Not supported.
 C_DestroyObject: Supported.
 C_GetObjectSize: Not supported.
 C_GetAttributeValue: Supported.
 C_SetAttributeValue: Supported.
 C_FindObjectsInit: Supported.
 C_FindObjects: Supported.
 C_FindObjectsFinal: Supported.
 C_EncryptInit: Supported.
 C_Encrypt: Supported.
 C_EncryptUpdate: Not supported.
 C_EncryptFinal: Not supported.
 C_DecryptInit: Supported.
 C_Decrypt: Supported.
 C_DecryptUpdate: Not supported.
 C_DecryptFinal: Not supported.
 C_DigestInit: Supported.
 C_Digest: Supported.
 C_DigestUpdate: Supported.
 C_DigestKey: Not supported.
 C_DigestFinal: Supported.
 C_SignInit: Supported.
```

```
 C_Sign: Supported.
 C_SignUpdate: Supported.
 C_SignFinal: Supported.
 C_SignRecoverInit: Not supported.
 C_SignRecover: Not supported.
 C_VerifyInit: Supported.
 C_Verify: Supported.
 C_VerifyUpdate: Supported.
 C_VerifyFinal: Supported.
 C_VerifyRecoverInit: Not supported.
 C_VerifyRecover: Not supported.
 C_DigestEncryptUpdate: Not supported.
 C_DecryptDigestUpdate: Not supported.
 C_SignEncryptUpdate: Not supported.
 C_DecryptVerifyUpdate: Not supported.
 C_GenerateKey: Not supported.
 C_GenerateKeyPair: Supported.
 C_WrapKey: Not supported.
 C_UnwrapKey: Not supported.
 C_DeriveKey: Not supported.
 C_SeedRandom: Supported.
 C_GenerateRandom: Supported.
 C_Finalize: Supported.
C_GetInfo ...
 Cryptoki Version: 2.20
 Manufacturer ID: Gemalto
 Flags: 0 (should be null)
 Library Description: Gemalto .NET PKCS11
 Library Version: 2.2
C_GetSlotInfo on slot 0
 Slot Description: SCM Microsystems Inc. SCR 3310 [CCID
    Interface] 00 00
 Manufacturer ID: Unknown
 Flags(7): CKF_TOKEN_PRESENT, CKF_REMOVABLE_DEVICE, CKF_HW_SLOT
    ,
 Hardware Version: 0.0
 Firmware Version: 0.0
C_GetTokenInfo on slot 0
 Label: .NET #CAEDD8E1ADC54808
 Manufacturer ID: Gemalto
 Model: .NET Card
 Serialnumber: CAEDD8E1ADC54808
 Flags (1036): CKF_LOGIN_REQUIRED, CKF_USER_PIN_INITIALIZED,
    CKF_TOKEN_INITIALIZED,
 Max sessions: 0
```

```
 Open sessions: 0
 Max R/W sessions: 0
 Open R/W session: 0
 Max. PIN length: 255
 Min. PIN length: 4
 Public memory: 18446744073709551615
 Free public memory: 18446744073709551615
 Private memory: 18446744073709551615
 Free private memory: 18446744073709551615
 Hardware Version: 0.0
 Firmware Version: 0.0
 Time:
C_GetMechanismList on slot 0
CKM_RSA_PKCS_KEY_PAIR_GEN, CKM_RSA_PKCS, CKM_RSA_X_509,
    CKM_MD5_RSA_PKCS, CKM_SHA1_RSA_PKCS, CKM_SHA256_RSA_PKCS,
    CKM_MD5, CKM_SHA_1, CKM_SHA256,
C_GetMechanismInfo on slot 0
 CKM_RSA_PKCS_KEY_PAIR_GEN: key size(512-2048)CKF_HW,
    CKF_GENERATE_KEY_PAIR,
 CKM_RSA_PKCS: key size(512-2048)CKF_HW, CKF_ENCRYPT,
    CKF_DECRYPT, CKF_SIGN, CKF_VERIFY,
 CKM_RSA_X_509: key size(512-2048)CKF_HW, CKF_ENCRYPT,
    CKF_DECRYPT, CKF_SIGN, CKF_VERIFY,
 CKM_MD5_RSA_PKCS: key size(512-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_SHA1_RSA_PKCS: key size(512-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_SHA256_RSA_PKCS: key size(512-2048)CKF_HW, CKF_SIGN,
    CKF_VERIFY,
 CKM_MD5: key size(0-0)CKF_DIGEST,
 CKM_SHA_1: key size(0-0)CKF_DIGEST,
 CKM_SHA256: key size(0-0)CKF_DIGEST,
Find objects on slot 0 as User
 Could find 0 objects of class 0
 Could find 0 objects of class 1
 Could find 0 objects of class 3
 Could find 0 objects of class 8
 Could find 0 objects of class 2
 Could find 0 objects of class 4
 Could find 0 objects of class 2147483648
Find public objects on slot 0
 Could find 0 objects of class 0
 Could find 0 objects of class 1
 Could find 0 objects of class 3
 Could find 0 objects of class 8
```

```
 Could find 0 objects of class 2
 Could find 0 objects of class 4
 Could find 0 objects of class 2147483648
Testing case: Minimal ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call GetInfo ... success
 call Finalize ... success
 ... case successful
Testing case: SlotInformation ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call GetSlotList with CK_FALSE and NULL_PTR ... success
 call GetSlotList with CK_FALSE ... success
 call GetSlotInfo on slot 0 ... success
 call GetSlotInfo on slot 1 ... success
 call GetSlotInfo on slot 2 ... success
 call GetSlotInfo on slot 3 ... success
 call GetSlotInfo on slot 4 ... success
 call Finalize ... success
 ... case successful
Testing case: TokenInformation ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_GetTokenInfo on slot 0 ... success
 call GetMechanismList with NULL_PTR on slot 0 ... success
 call GetMechanismList on slot 0 ... success
 call Finalize ... success
 ... case successful
Testing case: CompareMechanismLists ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call GetMechanismList with NULL_PTR on slot 0 ... success
 call GetMechanismList on slot 0 ... success
 iterate over all mechanisms and call C_GetMechanismInfo on
    slot 0 ... compared with MechanismList and the result is
    off by 0 mechanisms
```

```
 iterate over mechanismlist(C_GetMechanismList) and call
    C_GetMechanismInfo
 call Finalize ... success
 ... case successful
Testing case: SlotEvents ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call GetSlotInfo on slot 0 ... success
 A removable token was detected.
 call C_WaitForSlotEvent on slot 0 ...
   Please remove the token on slot 0.
 success
 call C_GetTokenInfo on slot 0 ... The token was removed.
 call C_WaitForSlotEvent on slot 0 ...
   Please insert the token on slot 0.
 success
 call C_GetSlotInfo on slot 0 ... the token was detected.
 call Finalize ... success
 ... case successful
Testing case: SingleSessionHandling ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION... failed as
    expected
 call C_GetSessionInfo on slot 0 with the previous session
    handle... failed as expected
 call C_CloseSession on slot 0 with the previous session handle
    ... failed as expected
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after logout)
```

```
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(session handle is invalid after close)
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(could verify state after logout)
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_GetSessionInfo on slot 0 with the previous session
    handle... success(session handle is invalid after close)
 call C_Finalize ... success
 ... case successful
Testing case: MultipleSessionHandling ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after login)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after logout)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after logout)
 call C_CloseAllSessions on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(session handle is invalid after close)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(session handle is invalid after close)
```

```
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 call C_Login on slot 0 as User ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after login)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after login)
 call C_Logout on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(could verify session state after logout)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(could verify session state after logout)
 call C_CloseAllSessions on slot 0 ... success
 call C_GetSessionInfo on slot 0 with session handle 1...
    success(session handle is invalid after close)
 call C_GetSessionInfo on slot 0 with session handle 2...
    success(session handle is invalid after close)
 call C_Finalize ... success
 ... case successful
Testing case: PreserveOperationState ...
 Check dependencies ...
  MISSING: C_GetOperationState
Testing case: AdminToken ...
 Check dependencies ...
 expected argument '-i' was not ommited
Testing case: ChangePin ...
 Check dependencies ...
 Dependencies are fine
 call C_Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_RW_SESSION &
    CKF_SERIAL_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_SetPIN on slot 0 as User ... success. New PIN is 0000
 call C_Logout on slot 0 ... success
 call C_Login on slot 0 as User with new PIN ... success
 call C_SetPIN on slot 0 as User set PIN back to 00000...
    success. New PIN is 00000
 call C_Logout on slot 0 ... success
 call C_CloseSession on slot 0 with the previous session handle
    ... success
 call C_Finalize ... success
```

```
 ... case successful
Testing case: ObjectHandling ...
 Check dependencies ...
  MISSING: _DestroyObject
Testing case: ObjectSearch ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_CreateObject with object class 0... success
 call C_CreateObject with object class 0... success
 call C_CreateObject with object class 0... success
 could create 3 data objects. So we should find 3 more than at
    the start
 Could find 3 objects of class 0
 call C_Logout on slot 0 ... success
 call C_CloseSession on slot 0... success
 call Finalize ... success
 ... case successful
Testing case: DeAndEncryptionSimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 De - and encrypting(CKM_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
C_Encrypt: CKR_OBJECT_HANDLE_INVALID
    failed
C_EncryptInit: CKR_OPERATION_ACTIVE
    failed
C_EncryptInit: CKR_OPERATION_ACTIVE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 De - and encrypting(CKM_RSA_X_509) ...
```

```
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
  encrypt + decrypt ...
C_Encrypt: CKR_OBJECT_HANDLE_INVALID
    failed
C_EncryptInit: CKR_OPERATION_ACTIVE
    failed
C_EncryptInit: CKR_OPERATION_ACTIVE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: DeAndEncryptionMultiple ...
 Check dependencies ...
  MISSING: C_EncryptUpdate
Testing case: DigestSimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 Digest:CKM_MD5 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
5730ebcb0077e6329c5a0534ddc865b8 <-
   eoatgormngwbvkmzsqlyfovnziqflkhrajkhzetmmrnkdalyswyxkumlcdqppxgp

 skipping C_DigestKey because it is not supported
 Digest:CKM_SHA_1 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
  call C_Digest on slot 0 ... success
9c65531de8c2f1393dc773a12e87208364325fe0 <-
   eoatgormngwbvkmzsqlyfovnziqflkhrajkhzetmmrnkdalyswyxkumlcdqppxgp

 skipping C_DigestKey because it is not supported
 Digest:CKM_SHA256 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_Digest on slot 0 with NULL_PTR ... success
```

```
  call C_Digest on slot 0 ... success
783575
   b4550b245a8cf229857e057d1d6f4053c7b72c0d23eebe10a2a4e3e4b0
   <-
   eoatgormngwbvkmzsqlyfovnziqflkhrajkhzetmmrnkdalyswyxkumlcdqppxgp

 skipping C_DigestKey because it is not supported
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: DigestMultiple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
    success
 Digest:CKM_MD5 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
0aec00ed0d65b689311dde94385f488d <-
   mrsdgrmwchwjgqdtcmruvakwfzwxjkbycvbimngoufacvfyzrpuoremyfkypuzny

 Digest:CKM_SHA_1 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
74f57c0f5480d732a4d3402of43ede24598b2124 <-
   mrsdgrmwchwjgqdtcmruvakwfzwxjkbycvbimngoufacvfyzrpuoremyfkypuzny

 Digest:CKM_SHA256 on slot 0 ...
  call C_DigestInit on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestUpdate on slot 0 ... success
  call C_DigestFinal on slot 0 with NULL_PTR ... success
  call C_DigestFinal on slot 0 ... success
22
   e7f769a6e8093497a68d1aca2b7679f5ff72a89316472f5fac86f0514079e0
```

```
    <-
    mrsdgrmwchwjgqdtcmruvakwfzwxjkbycvbimngoufacvfyzrpuoremyfkypuzny

 call Finalize ... success
 ... case successful
Testing case: SignAndVerifySimple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 Signing and Verify(CKM_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.729873 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
C_Sign: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_RSA_X_509) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.858916 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
C_Sign: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_MD5_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.715772 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.179676 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
```

```
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA1_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.836077 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.177484 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA256_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.719371 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.177082 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: SignAndVerifyMultiple ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 Signing and Verify(CKM_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.829949 sec.
C_VerifyUpdate: CKR_OBJECT_HANDLE_INVALID
    failed
C_SignUpdate: CKR_DATA_LEN_RANGE
```

```
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_RSA_X_509) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.671669 sec.
C_VerifyUpdate: CKR_OBJECT_HANDLE_INVALID
    failed
C_SignUpdate: CKR_DATA_LEN_RANGE
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_MD5_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.817034 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.183169 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA1_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.683934 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.181420 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 Signing and Verify(CKM_SHA256_RSA_PKCS) ...
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
```

```
 call C_Login as User ... success
   Could sign 16 CK_BYTEs in 0.808094 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
   Could sign 256 CK_BYTEs in 0.180878 sec.
C_Verify: CKR_OBJECT_HANDLE_INVALID
    failed
 call C_Logout ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: SignAndVerifyRecover ...
 Check dependencies ...
  MISSING: C_SignRecoverInit
Testing case: DualPurposeFunctions ...
 Check dependencies ...
  MISSING: C_DigestEncryptUpdate
Testing case: SingleKeyGeneration ...
 Check dependencies ...
  MISSING: C_GenerateKey
Testing case: KeyPairGeneration ...
 Check dependencies ...
 Dependencies are fine
 call Initialize ... success
 call C_GetSlotList with CK_TRUE and NULL_PTR ... success
 call C_GetSlotList with CK_TRUE ... success
 call C_OpenSession on slot 0 with CKF_SERIAL_SESSION |
    CKF_RW_SESSION... success
 call C_Login on slot 0 as User ... success
 call C_GenerateKeyPair(CKM_RSA_PKCS_KEY_PAIR_GEN) on slot 0
    ... success
 call C_Logout on slot 0 ... success
 call C_CloseSession with the previous session handle...
    success
 call Finalize ... success
 ... case successful
Testing case: WrapAndUnwrapKey ...
 Check dependencies ...
  MISSING: C_GenerateKey
Testing case: DeriveKey ...
 Check dependencies ...
  MISSING: C_GenerateKey
Testing case: GenerateRandom ...
 Check dependencies ...
```

```
Dependencies are fine
call Initialize ... success
call C_GetSlotList with CK_TRUE and NULL_PTR ... success
call C_GetSlotList with CK_TRUE ... success
call GetSlotInfo on slot 0 ... success
CKF_RNG is not set
call C_OpenSession on slot 0 with CKF_SERIAL_SESSION...
   success
call C_SeedRandom on slot 0 ...: CKR_RANDOM_SEED_NOT_SUPPORTED
call C_GenerateRandom(64 Bytes) 1000 times on slot 0 ...
 Could call C_GenerateRandom 1000 times with 64 bytes in
    5.926074 sec.
 append data to file rand_64.txt.
done.
call C_GenerateRandom(1024 Bytes) 1000 times on slot 0 ...
 Could call C_GenerateRandom 1000 times with 1024 bytes in
    5.753722 sec.
 append data to file rand_1024.txt.
done.
call C_GenerateRandom(8192 Bytes) 1000 times on slot 0 ...
 Could call C_GenerateRandom 1000 times with 8192 bytes in
    5.832173 sec.
 append data to file rand_8192.txt.
done.
call Finalize ... success
... case successful
```